Spring 2007

Detecting Compromised Nodes in Wireless Sensor Networks

Mary Lisa Mathews
Old Dominion University

DETECTING COMPROMISED NODES IN WIRELESS SENSOR NETWORKS

by

Mary Lisa Mathews
B.S. 2004, Drexel University

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirement for the Degree of

MASTER OF SCIENCE

COMPUTER ENGINEERING

OLD DOMINION UNIVERSITY
May 2007

Approved by:

Min Song (Director)

Lee A. Belfore (Member)

Frederic D. McKenzie (Member)

ABSTRACT

DETECTING COMPROMISED NODES IN WIRELESS SENSOR NETWORKS

Mary Lisa Mathews
Old Dominion University, 2007
Director: Dr. Min Song

While wireless sensor networks are proving to be a versatile tool, many of the applications in which they are utilized have sensitive data. Therefore, security is crucial in many of these applications. Once a sensor node has been compromised, the security of the network degrades quickly if measures are not taken to deal with this event. There have been many approaches researched to tackle the issue. In this thesis, an anomaly-based intrusion detection protocol is developed to detect compromised nodes in wireless sensor networks.

The proposed protocol is implemented after the sensors are deployed into the environment in which they will be used. They will start to learn the normal behavior of each of their neighbors with whom they communicate. All legitimate sensor nodes have the same code running on them. A compromised node that is present in the network is assumed to have different code running on it in order to cause some form of damage to the network. These malicious nodes are detected when one of its neighboring nodes identifies its behavior as deviating from what is expected, or in other words an anomaly. The base station is then contacted to confirm whether the suspected node is in fact compromised. If the base station concludes that the node is compromised, the rest of the network will be informed, and the appropriate actions will be taken. One of the unique features of the algorithm is that it is not only capable of sustaining security in wireless sensor networks, but handling the computing restraints as well as other limitations

characteristic of these systems. Extensive simulations are performed to verify the algorithm designed.

This Thesis is dedicated to my close family and friends, for their constant love and support.

ACKNOWLEDGEMENTS

I would like to thank Dr. Min Song for his continuous support and guidance on my research. I would also like to thank Dr. Lee Belfore and Dr. Frederic McKenzie for consenting to be on my thesis advisory committee.

I would also like to thank my parents for always being there for me and pushing me to strive for my best.

I would also like to thank Mr. Sachin Shetty for his help in understanding the TOSSIM simulator and getting over the many hurdles it presented.

TABLE OF CONTENTS

Page

LIST OF FIGURES

I. INTRODUCTION

Research into wireless sensor networks, as well as the applications that employ this technology that are already in place, show that they are a favorable method for solving problems or enhancing existing systems. They can be used for a wide range of functions from monitoring patients while not at the doctor's office to sensing environmental conditions such as the level of pollutants in a given area. Since many wireless sensor networks are utilized in applications where the data gathered is confidential, security has become a critical issue. However, finding an efficient solution to this issue is easier said than done for a variety of reasons.

1.1 General Wireless Sensor Network Characteristics

Wireless sensor networks are comprised of sensor nodes that are designed to gather information regarding environmental data such as light, temperature, sound, and pressure. These sensor nodes generally function using the same elements described below [13], [14]. To begin, there are the actual sensors which gather the data that represents the physical conditions being monitored once the network has been deployed. The sensor readings that are gathered periodically are sent to the processing unit that houses the data and program memory. This processing unit usually converts the data values into a human readable format if this is desired or needed. The operating system and other programs stored in the program memory dictate all operations of the sensor nodes while they are in use. While the power source of these devices normally comes in

the form of a battery, there has been research conducted into other sources such as solar cells. The wireless aspect of the communication in these sensor networks is usually achieved through a radio antenna, although some sensors have instituted infrared or laser communication schemes.

In many deployments, each node in the sensor network is given a unique identification number that is usually determined when the sensors are programmed. This means that this node id is part of the code in the program memory and is included in all outgoing packets sent to the rest of the network. This id is typically given by assigning numbers starting from either zero or one and incrementing by one for each node added to the network.

Implementing any form of security measure onto the sensor nodes require the use of resources that are already constrained in these networks [1], [2]. The sensor nodes are designed with the goals of being small, in order to be utilized in different scenarios, and relatively cheap so that many nodes can be deployed in the desired environment. This has led to these sensors having constraints in terms of low computation, memory, and power available. Therefore, any security method added to these networks will inevitably consume some of these resources [16].

A good portion of the memory is generally allotted for the code that runs on the sensors that instruct them on their sensing, communication, and other operational functions. The programs dictate what conditions to sense, when to sense them, and what computations to perform on the values gathered. The nodes also need to know when to construct packets, how to construct them, and who to send them to. This implies that the program determines who the nodes of the network communicate with in terms of sending and receiving information. Some applications allow all nodes to communicate with each

other while others require each node to keep a neighbor list of acceptable nodes with whom they are allowed to send and receive packets. Additional code that is required to implement any algorithm needs to fall within the range of the total provided memory minus the existing code memory already utilized. Increasing the resources on the sensor nodes is not a viable solution if the goals mentioned above or the general operating efficiency of the network are sacrificed.

1.2 General Security

The objective of any security method being utilized is to maintain authentication, secrecy, and data integrity within the network [2], [3], [18]. Authentication involves the receiver of a packet being able to validate that the alleged sender is in fact the real sender and that it is a valid node of the network. The node identification numbers assigned to each member of the network comes into play here. Secrecy (a.k.a. confidentiality) deals with making sure that the data sent is kept secret from those who should not have access to the information. Even if messages sent in the network are received by unintended parties, they should not be able to decipher the message contents. Data integrity ensures that the data received is the same as the data that was sent. Altered messages can come about by malicious parties modifying packets that are sent between nodes or as a result of distortion occurring from the wireless communication medium. Either way, these messages can produce adverse effects in the network, especially in real-time applications where any information received is acted upon accordingly. For instance, in a battlefield surveillance application, packet information that represents an approaching enemy would set off a drastic set of actions. If this information was incorrect, the result could be disastrous. Many applications benefit from implementing some form of data aggregation

to average out the values obtained from the sensors [2]. This way, the threats that any extreme values obtained from sensor readings, as well as any altered packets present in the network, pose to the reliability of the information gathered by the nodes is reduced.

Different types of attacks on wireless sensor networks focus on exploiting the resource constraints to cripple one of the three parameters listed above [3], [15]. An attacker can passively eavesdrop on the wireless communication occurring within the network. By doing so, any of the sensitive information that is being sensed by the nodes will be available to the listening party [14]. For a more active assault, a malicious party could inject false packets into the network that would be perceived as valid information by the other nodes [17]. This also ties up network resources that could have been used for legitimate packets. An attacker might also alter the contents of a valid packet, which undermines the authentication and data integrity of the network.

Most security algorithms employ some form of cryptography where data is encoded and then decoded by the base stations and sensor nodes of the wireless sensor network [4]. Cryptography allows for authentication, secrecy, and data integrity to be maintained within the network. TinySec is an example of a security protocol placed on wireless sensor networks that incorporates these traits [18]. However, the security of many of the algorithms degrades when one or more nodes have been compromised [5], [6]. A compromised node occurs when a once valid node of the network has been reprogrammed to perform some type of malicious activity. Previous security measures such as cryptography keys mostly likely will not work against them since the compromised nodes now have these keys, and the other nodes do not know they are compromised. They are still seen as valid nodes of the network. These nodes can perform many attacks that cause problems to the general network operation including

injecting false packets and modifying the contents of packets coming from other sensor nodes. The rest of the network would not be able to identify the compromised node, or intruder, from a valid node if there are not additional security measures included in the network.

Another assault that plagues wireless sensor networks is the selective forwarding attack [15]. In this attack, the adversary selectively forwards packets sent by other nodes in the network which results in lost information. For this to work, the malicious party needs to somehow include itself into the actual path of the packets being sent. If a compromised node has incorporated itself into the network and is undetected, it could easily perform this attack since the other nodes that consider it a neighbor would continue to send it packets. Information such as network updates vital to sensor network operation and packets containing sensor values would be prevented from propagating through the network correctly. This causes damage to the traffic flow of the network as well.

1.3 Intrusion Detection Systems

An intrusion detection system (IDS), whose function is to detect attacks that exploit the vulnerabilities or flaws within a given network, could be utilized in this situation [7]. They are generally classified into two main types: misuse intrusion detection and anomaly-based intrusion detection [12]. Both kinds strive for the same characteristics of a 100% attack detection rate as well as a 0% false positive rate. The 100% attack detection implies locating and stopping any attacks that occur while the network is up and running. A false positive occurs when a legitimate node is identified as an intruder by the other nodes. For obvious reasons, this is detrimental to the integrity of the network and decreases the competency measurement of the detection system.

Misuse intrusion detection systems work under the concept that the attacks that plague a network exhibit certain unique characteristics that can form a signature for said attack [12]. The individual attacks are introduced onto the network and studied in order to look for patterns with which to identify the attack. While the network is deployed, it is constantly being monitored for activity that matches any of the known signatures. If a match is found, appropriate action is taken to deal with the intruder that has been identified. The problem with this type of IDS is that unknown attacks can pass through the network undetected.

In anomaly-based intrusion detection systems, there is an assumption that the intruder's behavior deviates from the normal network behavior [12]. In this type of IDS, each sensor node will be monitoring its neighbors to keep track of the normal behavior for a given set of parameters. The nodes develop profiles for each of the nodes with whom it communicates to determine what is acceptable in terms of communication in the form of packets sent and received. Any node that strays from its standard actions will trigger an alarm in its neighbors. The disadvantage of this type of IDS is that there is generally a high false positive rate. Also, the computation involved with figuring out whether each neighboring node has deviated from its acceptable behavior along with updating the profiles has the potential to overtax the network resources.

1.4 Combining Two Methods to Detect Compromised Nodes

In this thesis, two methods for detecting compromised nodes were combined to create a new algorithm that improved on them by overcoming their limitations. A brief summary of the methods as well as the solution to their limitations is described below. A more detailed summary of the methods is provided in the related work section for each.

1.4.1 Network/Neighbor Stability Based Anomaly Detection

Onat and Miri developed an anomaly-based intrusion detection system in which each node created profiles of acceptable behavior for each of its neighbors based off two parameters which are packet receive power and packet arrival rate [10]. Each node was programmed to store a packet buffer that would determine the threshold values which are the highest and lowest acceptable values for these parameters. This buffer was to be maintained and updated for the entire lifetime of the network. A packet is labeled as anomalous when the observed parameter value does not fall within the range allowed by the threshold values. An intrusion buffer is kept for each neighbor as well. This buffer stores a preset number of consecutive packets that do not fall within the expected range. If a packet that falls within the desired range is received before the preset number is reached, the intrusion buffer is emptied. However, an intruder is identified when its intrusion buffer is filled.

In the algorithm designed for in this thesis, an intrusion detection system similar to the one described above was implemented, although there were some limitations that needed to be overcome. To begin with, there was a large amount of space needed in memory for the two buffers to be able to identify the compromised nodes. The packet buffer stored 1000 entries while the intrusion buffers stored 100 when dealing with the packet arrival rate. In this thesis, the packet arrival time is the parameter used to generate the profiles of acceptable behavior. The algorithm described in the paper was designed to work on large scale networks. However, small scale networks are used during the simulation runs for this thesis. The packet buffer implemented here kept 10 entries, while the intrusion buffer kept 1. This reduction is mainly due to the use of a trusted entity which will be described in the next section and the network size. The trusted entity,

which is a base station in this thesis, also eliminates the occurrence of any false positives. The algorithm created here would be applicable to large scale networks if certain changes are implemented. The packet buffer would be increased to 100 entries, and the number of base stations utilized would increase as a function of the network size.

The authors mentioned that in most types of anomaly-based intrusion detection systems, each node needs to hear that a fixed number of other nodes suspect the intruder as well to confirm that the node is indeed compromised. While they left this for future work, this had been dealt with here. There are four packets transmitted between the trusted entity and other nodes of the network to identify a compromised node and broadcast its decision to the rest of the network. Once a sensor node hears this message, it will cease to communicate with the identified intruder.

1.4.2 SWATT: Software-based ATTestation for Embedded Devices

The creators of SWATT revealed that code attestation can proficiently identify intruders in wireless sensor networks [11]. Their technique was based on the perception that a compromised node will have different code running on it and stored in its memory compared to a valid node of the network. This is a viable claim considering that if a node is truly compromised, it has to do something different compared to the legitimate nodes in the network. Given that the program being executed on the nodes controls their operation, malicious behavior would be written in the same memory contents where the program is stored. This principle is incorporated into the design of this thesis as well.

A verifier is utilized in SWATT to identify intruders. It was assumed that the verifier has a copy of the code running on the sensor nodes of the network. The verifier instructs a node to perform a checksum over the contents of memory while performing

the checksum itself over the copy it has on hand. If the response of the requested node does not come in time or is incorrect, an intruder has been found.

The main limitation with directly implementing their work in the algorithm designed for this thesis is there was no mention of a way to detect a compromised node. The simulations involved checking nodes that were already known to have altered code. The solution implemented here is an anomaly-based intrusion detection system mentioned in the previous section.

1.5 Challenges and Contributions of Thesis

This thesis presents an anomaly-based intrusion detection system that deals with the threat imposed by the selective forwarding attack on wireless sensor networks. When dealing with this type of environment, the resource constraints that are characteristic of these networks was taken into consideration. The additional code needed to implement the proposed algorithm needed to work on top of the existing code that manages all sensor operational activities. It was crucial that the additional memory space required for each node to store profiles containing acceptable behavior information regarding its neighbors be kept to a minimum. Any computations involved with determining whether a node deviated from its normal behavior was carefully weighed for usefulness and necessity. These computations are to be performed for every single incoming packet that a node receives from each node it considers a neighbor.

Apart from the resource limitations present, there was also the wireless communication aspect to consider. The fact that the all traffic within the network uses this medium implies a higher number of packets lost and a higher number of packets dropped. This affects the normal behavior profiles stored for the neighboring nodes.

There were also the questions of how to efficiently determine a node as compromised and how to propagate the identification of a compromised node to the rest of the nodes.

II. RELATED WORK

2.1 Proving the Necessity of Security

Wireless sensor networks have already been exploited in a variety of applications. While some of the people using these sensors have protected their networks with different measures such as cryptography keys, there may be others that still do not realize the importance of security or the multitude of threats that these networks are susceptible to. Hartung, Balasalle, and Han proved the high level of vulnerability that is an intrinsic part of these networks [6].

The main contribution the authors provided in the paper was to demonstrate the ease with which an outsider can view as well as alter the code running on a sensor node once it has been obtained. Their experiment was conducted on the Crossbow Mica2 sensors. The Crossbow sensor platform, as well as most other sensor platforms, has a programming interface board that is used to write the programs onto memory. This board allows the sensor code to be changed to meet the specifications of the user instead of having sensor nodes with static code that cannot be changed. UISP, a free downloadable tool that one can use to interact with microcontrollers, was employed and easily copied the program code as well as the contents of EEPROM. The EEPROM is where the where the values read in from the actual sensors are stored. This code was then converted into assembly language, which allowed the authors to view any predefined keys and routing protocols that were present.

The second stage of the experiment revealed additional information that an attacker can effortlessly discover once a sensor node is obtained. An AVR JTAG interface was utilized in this part, which provides a means to manipulate signal levels,

programming, and On Chip Debugging (OCD) [6]. This device allowed the authors to gain access to the contents of the program, EEPROM, and the SRAM. It was noted that the SRAM is thought to be a secure location in which to keep secretive information. Obviously this theory was proven false by their experiment.

2.2 Location Based Anomaly Detection

2.2.1 LAD Scheme

Many wireless sensor networks utilize a GPS system to gather data regarding the location of the sensor nodes. In large sensor networks, providing each node with GPS capability might be too expensive; instead, many applications resolve this issue by using beacon nodes that have a GPS receiver. These beacon nodes will know their own location and send out beacon packets to the rest of the network that contain their location information. The sensor nodes that receive these beacons use their information to figure out their own location in the deployed network. The Local Anomaly Detection (LAD) scheme was devised to deal with the situation where there are compromised nodes that attack the localization schemes that are needed when implementing beacon nodes [8].

Their method utilizes knowledge regarding deployment for estimating sensor positions along with knowing which neighboring node belongs to which group. After deployment of the sensor network, a localization phase takes place where the nodes calculate their locations using the beacon nodes. Their detection scheme takes place in the next phase, which they called the detection phase, where the sensors try to analyze whether the locations they computed in the previous phase are indeed accurate. An anomaly is suspected when the variation between the expected location and the actual location of one of the nodes is too great.

There were three metrics designed by the authors to use in their anomaly-based intrusion detection scheme. They are all based around the fact that the sensors in their networks are grouped together by their deployment locations. The sensors will send their group id to neighboring nodes of the network. The first metric described is the Difference Metric (DM) which compares the observed number of nodes in its group recorded from the broadcast packets to the expected number of nodes that is the result of a calculation. An anomaly is indicated if the difference between the two numbers exceeds a preset number. The second metric, coined the Add-all Metric (AM), involves looking at the observed and expected number of neighbors in each group and performing a union function on them. The result is again compared to a threshold value, which if exceeded flags an anomaly. Last is the Probability Metric (PM) which involves each node calculating the probability that the observed number of members in each group is possible when considering its own estimated location. A threshold value is used in this metric as well. The threshold values of all three metrics was determined by running simulations of a wireless sensor network, collecting the data gathered, and performing the various computations desired for each metric.

The simulations indicated that the LAD scheme can indeed detect compromised nodes. The chances of detection are increased if the compromised nodes attack with a higher degree of damage. Obviously, the higher the number of compromised nodes there are in the network, the lower the detection rate becomes. The authors of the paper observed that the DM performed better than the other two metrics created, and they mainly used this one in their simulation runs to check the performance of their designed intrusion detection system. Although it was mentioned that compromised beacon nodes present a problem, they were not specifically addressed in the simulations or the LAD

scheme. What might be interesting to study here is if the positioning of the compromised beacon nodes affects the detection rate.

2.2.2 Detecting Malicious Beacon Nodes in LAD

The creators of LAD later published a paper presenting a way to detect malicious beacon nodes [9]. They reasoned that it would be difficult for a compromised beacon node to get away with sending beacon signals with the wrong location information. This is because the malicious beacon node location information and the beacon signal will both have to be falsified. All beacon nodes of the network can estimate the location of other beacon nodes based off received beacon signals.

Beacon nodes in the network are given a set of node id's and keys that allows them to communicate with the other beacon nodes of the network while appearing to be a non-beacon node. This was done to help decrease the chance that a compromised beacon node can determine when another beacon node is requesting information from it instead of a sensor node of the network. Compromised nodes are detected when a valid beacon node gets a beacon signal from a malicious beacon node whose estimated location based off the beacon signal is different from the location given by the beacon signal. Attacks using locally replayed beacon signals are discovered since it is difficult for the compromised beacon node to achieve the expected round trip time for communication between neighboring nodes. The authors also proposed a method to have a base station revoke malicious beacon nodes once there are enough alerts from other beacon nodes regarding the suspected node.

Simulation results showed that malicious beacon nodes are identified at a higher rate when they inject more malicious beacon signals. The results did show that their

algorithm detected malicious beacon signals and the base station was able to identify and stop communication with the compromised nodes. Replayed beacon signals were also detected which helped to decrease the number of false positives that occurred during network operation.

2.3 Network/Neighbor Stability Based Anomaly Detection

Onat and Miri developed an anomaly-based intrusion detection system by exploiting certain characteristics of the sensor nodes, namely their stable neighborhood information [10]. The sensor network they set up took the communication to be a many-to-one arrangement, which is where the sensor nodes send their information to a single or fixed destination along paths that are more or less stable. Therefore, the HELLO flood packets that nodes use to identify their neighbors would not be needed throughout the lifetime of the network.

There were many assumptions made by the authors when designing their algorithm and running their simulations. It was noted in the paper that new nodes did not appear in the network after initial deployment and that the nodes were not mobile. Every node in the network had the ability to distinctively identify its neighboring nodes. Also, there were thought to be no changes in transmission power levels. Each node used the same hardware and was managed by the same protocol stack. However, the clock running on one node was not presumed to be synchronized with those of the other nodes.

Given the stability of the network that was assumed, the sensors should know what to expect from their neighbors. To further exploit this concept, two parameters pertaining to their neighbors were chosen for each sensor node to store and maintain.

The parameters selected were the packet arrival rate in units of packets per unit time and the average receive power in units of dBm.

A buffer containing a predetermined number of packets is maintained in this algorithm. The packets coming into buffer before it has been filled are used to calculate the range of acceptable values with respect to the arrival rate and receive power for subsequent packets. After the buffer is filled, any arriving packets that match the criteria for admissible packets are used to update the acceptable range of values. This way, the older values are discarded to make room for the new. The criteria, when looking at the receive power for each packet that comes in, is to check whether it falls within the minimum and maximum receive power values of packets already in the packet buffer. For the packet arrival rate portion of the algorithm, two rates were maintained. The first was the rate at which the previous packets arrived, and the second is the rate at which the packets including the new one arrived. A compromised node is found if the ratio of these two rates exceeds a predetermined threshold value.

When the packet values for either of the specified parameters do not conform to the profile set for that particular node, the packet is added to an intrusion buffer. This intrusion buffer is used to store packets whose values do not fall within the expected ranges. After a preset number of sequential packets deviate from what is expected, the node sending these packets is labeled as an intruder [10]. If the node in question sends a series of anomalous packets and then one that is acceptable, the intrusion buffer is emptied.

The authors performed many simulations to check the credibility of their proposed intrusion detection system. As predicted, higher levels of variation between a node's transmit powers before and after it is compromised leads to a higher detection

probability and a decrease in the average detection time. This implies that if the malicious node is able to mimic the original node's transmit power level, it would be difficult to detect its presence. Regarding the packet arrival rates, detecting an intruder is more likely to occur when the threshold value used to determine the allowable range is lower. However, as the threshold value decreases, the number of false alarms was seen to increase. In other words, when the acceptable range of values is smaller, more nodes in general are flagged as intruders. This includes compromised nodes as well as legitimate nodes of the network.

It was observed that the probability of labeling a valid node as an intruder decreases as the number of packets stored in the intrusion buffer increases when using either one of the parameters measured for anomalies. This is important since most algorithms do not provide a method for nodes that are falsely accused of being compromised to reinstate themselves into the network later. Preventing these false alarms is important, especially in anomaly-based intrusion detection systems where they tend to be high. However, the opposite is true for identifying the compromised nodes when measuring transmit power anomalies, meaning that as the number of packets stored in the intrusion buffers increases, the probability of detecting the intruders decreases. This is because the time to detect the intruder inevitably increases seeing as how more packets are needed to fill up the buffer. During this time, the questionable node might send an acceptable packet that cleans out the intrusion buffer that is being kept for it.

The authors mentioned that in anomaly-based intrusion detection systems, a general consensus regarding the presence of the intruder is needed to decrease the number of false alarms and increase the chance of detecting the compromised nodes. Once a node suspects a compromised node is in the network, it will alert its neighbors by

sending them a packet identifying the possible intruder. When a node hears this alert message, it checks the unique node number contained in the packet to how many times this particular one has been suspected. After the node in question has been suspected for a predetermined number of times by neighboring nodes, it is identified as an intruder. This information can then be sent to the rest of the network so the attacker is now revealed to those who did not know of the intruder's presence.

2.4 SWATT: Software-based ATTestation for Embedded Devices

A different approach to detecting compromised nodes involves using code attestation to validate the actual program code running on the sensor nodes. Hardware-based methods of attestation exist where a secure coprocessor is utilized to check the memory contents of the embedded device in question. SoftWare-based ATTestation technique (SWATT) is a code attestation algorithm that is executed solely through software means [11].

Their technique was designed with the intention of creating a method to externally verify the code running on embedded devices. A trusted verifier is the key component in achieving this goal of their algorithm. The malicious node will contain at least one line of code that is different from the expected code running on normal sensors. The verifier has a copy of the memory contents residing in uncompromised nodes. The verifier sends a "challenge" to the node, which it uses as the input to a pseudo-random generator to create random memory addresses [11]. A checksum is performed in the device on each of these memory addresses. The verifier runs the same verification procedure locally to compute the expected value of the checksum. This expected value is compared to the value returned by the node in question.

A compromised node that has altered the memory contents would have to discern whether each memory location created by the pseudo-random generator has been changed. For the proposed SWATT method to perform well enough to be used in wireless sensor networks, the additional time needed to perform this check and run the verification procedure should be noticeable to the verifier. The authors chose to run their experiments on a simulator contained in AVR studio version 4.0. Their results showed that difference in time to compute the checksum becomes more prominent as the number of memory locations accessed increases.

The authors made the point that attestation done in the software is more suitable for sensor networks compared to a hardware-based method. This is due to the foresight that changes made to the hardware of the sensor nodes would most likely increase the production costs. The more nodes used in the sensor network, the more expensive it would be to implement code attestation in the hardware. Also, altering the software of the existing sensors to implement code attestation would be much easier than altering the hardware. Changing the software simply involves updating the code already running on the sensors to include the attestation program.

III. ALGORITHM FOR DETECTING COMPROMISED NODES

3.1 The Main Idea

The algorithm that was designed in this thesis for detecting compromised nodes in wireless sensor networks takes the general anomaly-based intrusion detection system and combines it with the event-driven property of sensor nodes. Selective forwarding attacks, which these networks are susceptible to, are thwarted in this algorithm.

Every node in the network learns when it can expect to receive packets from each of its neighbors. The compromised nodes in this algorithm perform the selective forwarding attack, meaning they do not send packets when they are supposed to. When the neighbors of this compromised node discover that they are not receiving packets from it as they should, they suspect that the node is an intruder. The various assumptions and the specifics of the design are described in detail below.

3.2 The Algorithm

There were many assumptions made to create the algorithm designed in this thesis. The sensor network in each simulation run would be of a fixed size, which ensures that malicious parties that add their own node into the network cannot participate in the communication amongst the valid nodes since they do not have a valid network id. The base station, assigned to node id of '0', was known to be a trusted source; therefore, any packets received by the node '0' would not be stored in the arrival_time_buffer[N] since there would be no anomalies to observe. Basically this means that the base station would never be compromised, so there was no reason to waste memory space for it in a packet buffer.

While some algorithms that use a trusted party for security or other purposes place more than one trusted entity in their wireless sensor network, there will only be one in this algorithm. The base station should have more computational and memory resources that allow it to take on the added tasks and communication that comes with its job. This allowed for the assumption that only one was needed, and adding additional base stations would be a waste of resources given the smaller size of the networks used in the simulations.

Each node in the network is to maintain an arrival_time_buffer[N] for each of its predetermined N number of neighbors. The number of neighbors will be determined based on a function of the network size seeing as how a smaller sized network would need fewer neighbors per node for communication purposed than a larger sized network. Also, it would take more time and resources to maintain a larger buffer, so a smaller size buffer that provides a balance between size and functionality is desired. The values stored in the buffers will be used to calculate the threshold values for packet arrival times for each neighboring node once a preset number of packets, max_buffer_packets, have arrived from said party. These threshold values basically provide a range of time in seconds during which the next packet for the particular node is expected to arrive. In other words, there is a high_value and a low_value in units of seconds that is stored for each neighbor, and each incoming packet needs to fall within these two values.

It is important to note than an assumption was made that compromised nodes will not be present in the sensor network during the neighbor discovery phase that takes place after deployment. The nodes need time to recognize the normal behavior of their neighbors. If their neighbors were already compromised, what is thought to be normal behavior would actually be abnormal and would deter efforts to detect intruders through

the discovery of anomalies. The compromised node created in this algorithm will perform the selective forwarding attack once it has become compromised. It will still communicate with the same nodes as it did when it was valid. The difference is now the node does not send packets on as it should. Rather, it does so randomly.

During this neighbor discovery phase, or initialization phase, every sensor node in the network is to receive a certain number of packets, max_buffer_packets, from each of its neighboring nodes. Each packet's arrival_time that comes in from the neighbor before the preset number is reached is checked against the high and low values recorded for the neighbor's node id. If the arrival time is higher than the high_value or lower than the low_value, that particular value is then updated.

It is important to note that the arrival_time is computed by the node itself and not taken from the packet timestamp. Each node knows the last time when it has received a packet from every single node it considers a neighbor. The clocks running on individual sensor nodes do not need to be synchronized given that its main function is to provide a means for determining the difference between receiving two incoming packets, which is stored in arrival_time_buffer[N]. These values are relative only to the node performing the calculations, meaning it does not need to look at those values calculated on other nodes.

Once the max_buffer_packets that were needed to compute these threshold values has been received, the arrival_time of each packet subsequent that comes in from that specific neighbor is checked to see if it has arrived in an acceptable time frame with respect to the last packet it sent. If the packet does not fall in the desired range, an ALERT message is sent to the base station that contains the node id of the suspected node, $node\_id_{compr}$ as wells as its own node id, $node\_id_{s}$.

Every node operating in the wireless sensor network, with the exception of the base station, is to keep a packet transmission time buffer, transmission_time_buffer[X]. This buffer maintained on each node stores the number of timer events, X, that have passed since that node has sent a packet, regardless of who the destination node is. Only a small number of entries need to be stored in this buffer for it to serve its purpose.

Upon receipt of an ALERT message, the base station will proceed to send a message, REQUESTION_TRANSMISSION_TIMES(), to the $node\_id_{compr}$ asking it for its transmission_time_buffer[X]. If the differences in these transmission times are not consistent, the base station is assured that the node is indeed compromised. The base station will then proceed to send out a broadcast message, COMPROMISED_NODE_FOUND($node\_id_{compr}$), to all sensor nodes of the network informing them of the presence of the compromised node. Upon receipt of this message, the nodes will clear out the values stored for $node\_id_{compr}$. Any packet coming from the compromised node will be ignored from this point onwards. An integral function the base station is assumed to have is the ability to communicate directly with each node of the network. That way, the exchange between the base station and the nodes will not be hampered by packets that are dropped when it is necessary for other nodes to pass on their packets.

The decision was made to update the high_value and low_value contained in the arrival_time_buffer[N] of the sensor nodes to keep it fresh. This was accomplished by having each node to store the values that fall outside the desired range in the arrival_time_buffer[N] when it sends the ALERT message to the base station. If the base station determines the suspected intruder is in fact a valid node of the network, it will send a message back to the node that sent the ALERT, $node\_id_s$, informing of its

findings. This node will then update its high_value or low_value accordingly for the appropriate node, $node\_id_{valid}$, in its arrival_time_buffer[N].

Due to the fact that dropped packets are prevalent in wireless sensor network, the base station was equipped with a buffer, alert_buffer[Y], in which to keep the last Y ALERT messages it receives. When the timer event has fired on the base station, the next node id, node_id, indicated in the alert messages buffer will be sent a request for its packet transmission times. This node id is determined based off a variable that contains the location of the next entry to be sent. This variable is updated so that the node ids are contacted based off the order in which the ALERT messages came in.

Figures 1 through 4 shown below provide the general concepts of the algorithm designed in this thesis through flow charts and pseudocode.

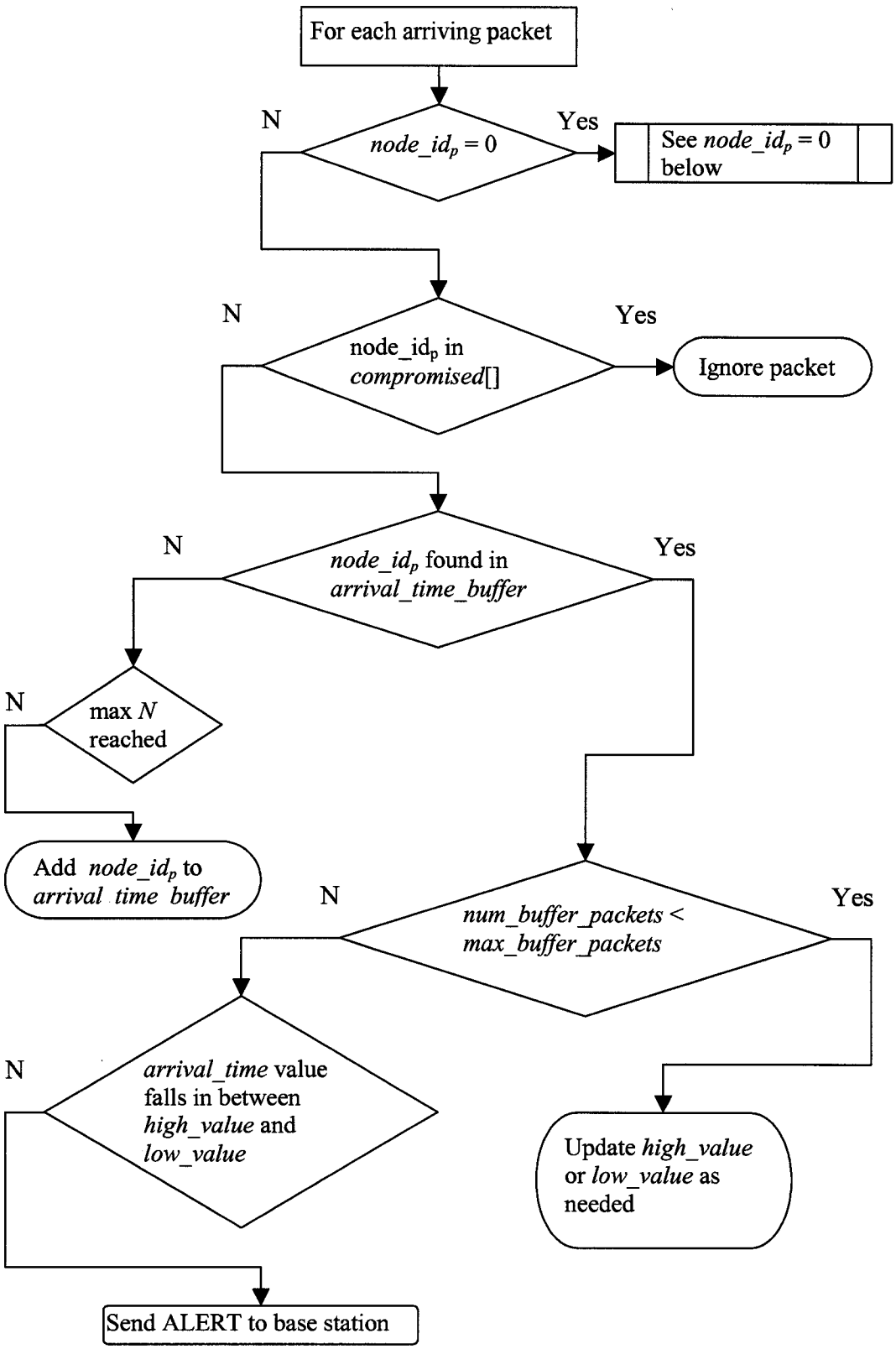

Figure 1. Flow chart on general sensor operation in algorithm.

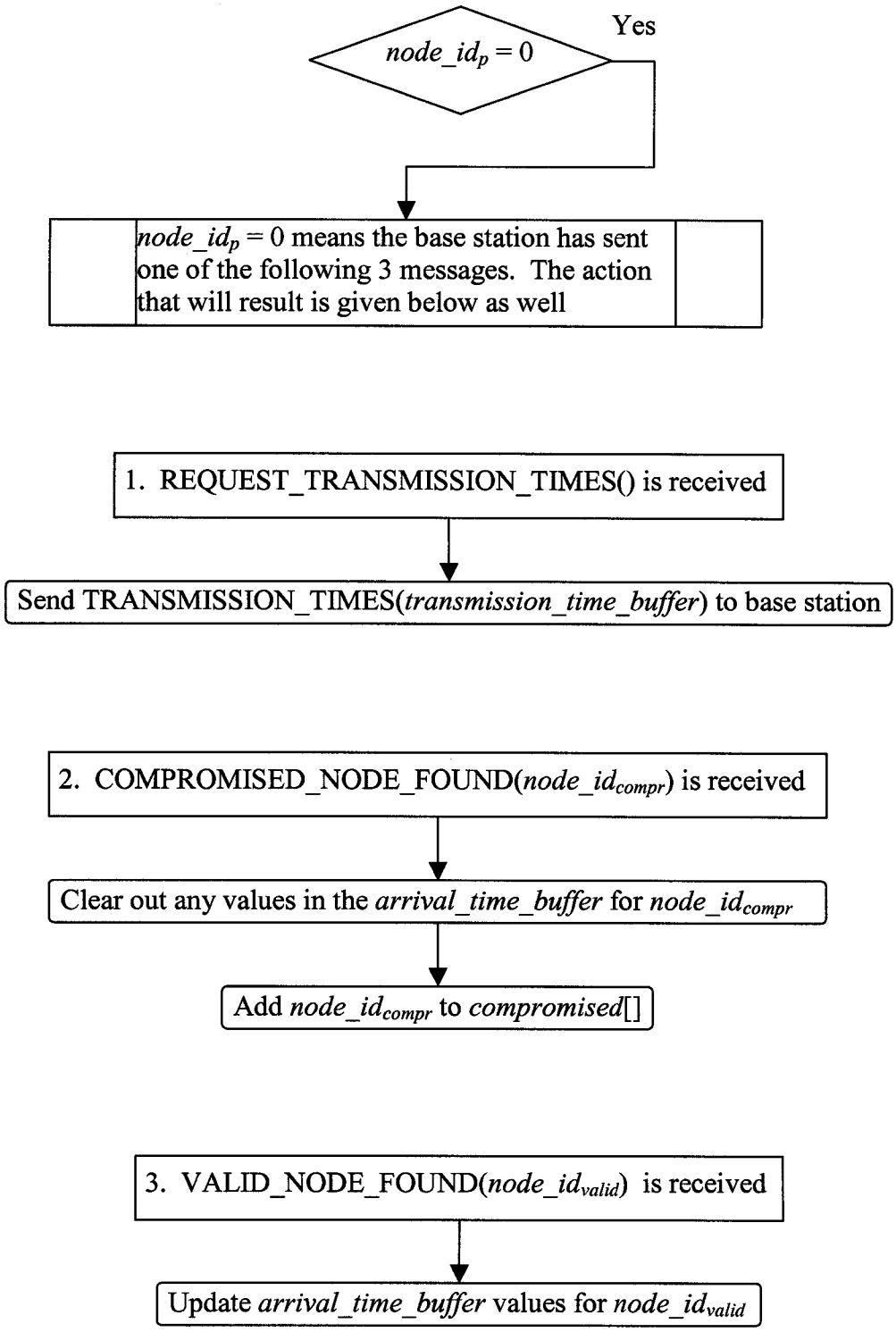

Figure 2. Flow chart for how sensors deal with messages from base station.

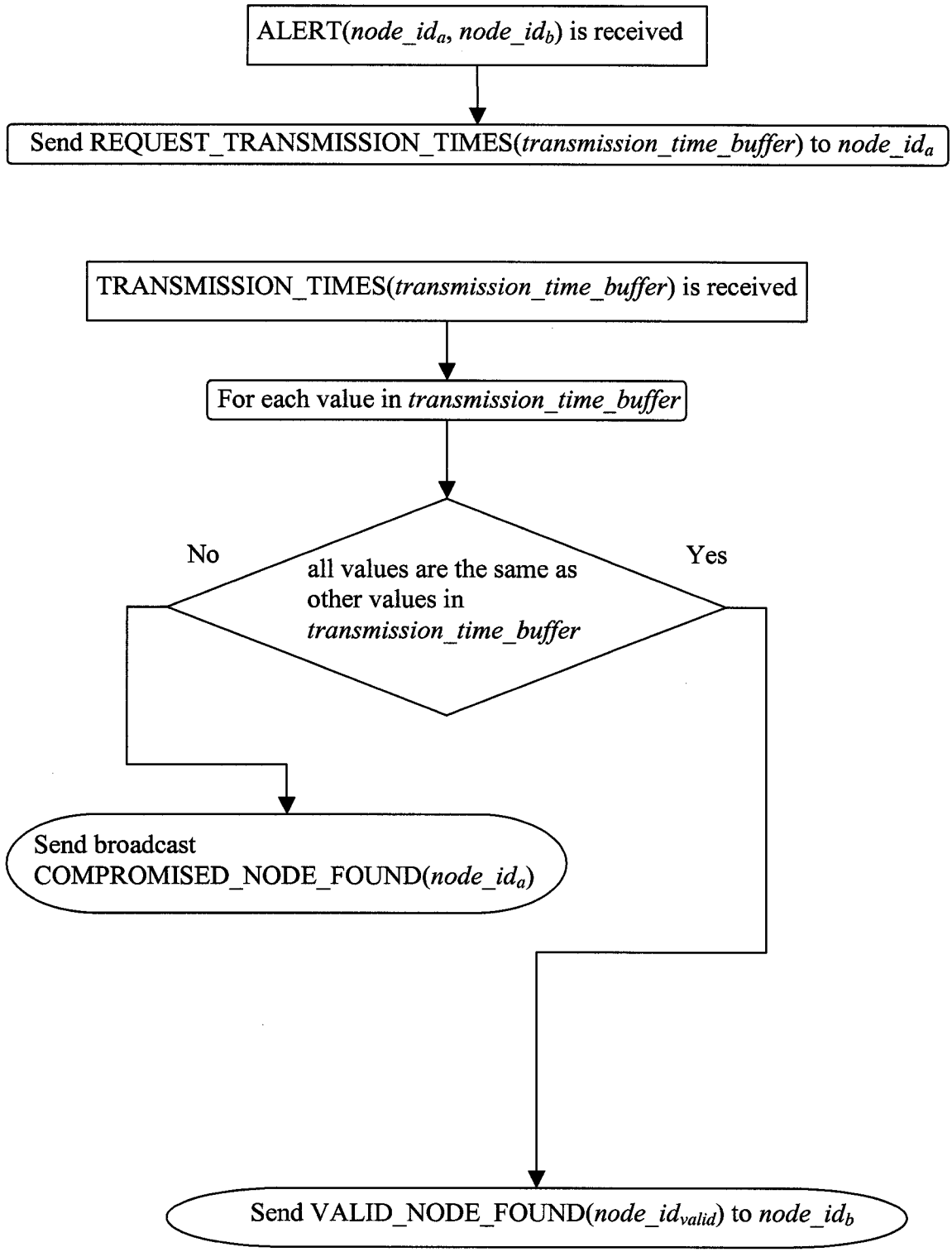

Figure 3. Flow chart on general base station operation in algorithm.

Pseudocode

Notation

For Sensor Nodes

node_id$_p$	node id of received packet
arrival_time	arrival time of packet based on system clock
arrival_time_buffer[N]	buffer that stores information regarding packets received for N nodes
max_buffer_packets	maximum number of packets to use for arrival_time_buffer
num_buffer_packets	number of packets already used for arrival_time_buffer[N]
high_value	highest acceptable packet arrival time
low_value	lowest acceptable packet arrival time
transmission_time_buffer[X]	buffer that stores last X transmission times
node_id$_s$	node id of sensor
node_id$_{compr}$	node id of compromised nodes identified by base station
node_id$_{valid}$	node id of valid node identified by base station
compromised[]	buffer that stores node ids of compromised nodes identified by base station

For Base Station

node_id$_a$	node id of suspected compromised node
node_id$_b$	node id of node that sent ALERT
alert_buffer[Y]	buffer that stores the node_id$_b$ of last Y ALERT messages received

1) Sensor Code

```
On {arrival of} packet
    If (node_id_p != 0) AND (node_id_p NOT in compromised[])
        If node_id_p already in arrival_time_buffer
            If num_buffer_packets < max_buffer_packets
                If arrival_time > high_value
                    high_value = arrival_time
                Else If arrival_time < low_value
                    low_value = arrival_time
                End If
            Else
                If (arrival_time > high_value) OR (arrival_time < low_value)
                    Send ALERT(node_id_p, node_id_s) to base station
                End If
            End If
        Else
            If max N not reached
                Add node_id_p to arrival_time_buffer
            End If
        End If
    End If
```

Figure 4. Pseudocode to be implemented.

```
On {arrival of} REQUEST_TRANSMISSION_TIMES()
```

Send TRANSMISSION_TIMES(*transmission_time_buffer*) to base station

On {arrival of} COMPROMISED_NODE_FOUND($node\_id_{compr}$)
Clear out any values in the arrival_time_buffer for $node\_id_{compr}$
Add $node\_id_{compr}$ to *compromised[]*

On {arrival of} VALID_NODE_FOUND($node\_id_{valid}$)
Update *transmission_time _buffer* values for $node\_id_{valid}$

2) Base Station Code

On {arrival of} ALERT($node\_id_a$, $node\_id_b$)
Send REQUESTION_TRANSMISSION_TIMES() to $node\_id_a$

On {arrival of} TRANSMISSION_TIMES(*transmission_time _buffer*)
For i < size of *packet_ buffer*
If *transmission_time _buffer*[i] != *transmission_time _buffer*[i + 1]
Compromised node identified
Send broadcast COMPROMISED_NODE_FOUND($node\_id_a$)
End
End For
If compromised node not identified
Send VALID_NODE_FOUND($node\_id_{valid}$) to $node\_id_b$

Figure 4. continued.

3.3 Type of Attack Thwarted by Algorithm

In the selective forwarding attack that plagues wireless sensor networks, packets that should be sent by a compromised node if it was still a valid node are selectively dropped. The compromised nodes purposely eliciting malicious behavior in this thesis will only perform this type of attack. They will be valid nodes of the network that are set to be compromised after a certain amount of time. The other nodes of the network that consider this node a neighbor should realize if there is a noticeable time difference between incoming packets. When this happens, they will send the ALERT message to the base station which will in turn catch the differences in the transmission time buffer of the compromised node. The selective forwarding attack is defeated when the other nodes of the network received the broadcast message sent by the base station informing them to cease communication with the malicious party.

3.4 Features of the Algorithm

This thesis relies on the fact that many wireless sensor networks are event driven, more specifically around a timer. In other words, the functions performed by the sensor nodes are dictated by a timer. These functions are those contained in the program running on the sensors that dictates all of their operations throughout the network lifetime, which includes the transmission of packets. This is the reason behind programming each sensor to keep a buffer containing the packet transmission times for all the packets it sends. When sensors are instructed by the base station to send transmission_time_buffer[X], the differences in those times will be calculated. If the differences do not conform to the timer specifications, meaning a packet was not created in the specified number of seconds, a node is deemed compromised.

In this algorithm, the base station, a trusted entity, is instructed to verify whether a suspected node is indeed compromised instead of leaving the task to the other sensor nodes in the network. Multiple compromised nodes might hinder the efficiency of finding intruders since neighboring nodes are the ones to verify the claims. If the situation arises in which neighbors are compromised as well, this presents a problem, especially if they are dropping packets. In this case, a valid node might not get the verification on its claim of a compromised node if the required number of nodes does not respond. The base station used in this algorithm is capable of more accurately validating whether a node is in fact an intruder. Also, if a node receives a message from the base station identifying a compromised node, it can immediately cease communication with the specified node. It does not need to hear similar messages from other nodes which would reduce the time needed to stop the compromised node from creating further havoc in the wireless sensor network.

Since the inter-nodal communication of having other nodes verifying intruders is no longer needed, a portion of the resource consumption in terms of battery usage, computations performed, and radio communication is reduced. The radio communication is especially important since congestion is often a problem in wireless sensor networks. In many intrusion based systems, a large amount of packet information would need to be stored to implement packet arrival rate based anomaly detection. This is not necessary for the algorithm designed in this thesis to be effective.

IV. SIMULATOR DEVELOPMENT

4.1 Simulator Introduction

The TinyOS simulator TOSSIM was chosen to implement the designed protocol for detecting compromised nodes in a wireless sensor network. This simulator has a smaller learning curve compared to more complex simulators such as NS2. TOSSIM was run in a Windows environment, and the TinyViz program available through TinyOS served as a GUI for TOSSIM. This GUI allows for the end user to enable various settings such as viewing debug messages, positioning the nodes in desired positions, and viewing the radio communication between nodes as it happens. TOSSIM was capable of simulating the interaction of networks with hundreds of sensor nodes.

The TinyViz program that was provided was very simple in nature. It was designed to simulate the communication occurring between sensors within a wireless sensor network. The sensor nodes simply sent messages once a timer event was fired if it knew its neighbors or broadcast messages to the whole network if it did not. The sender's node id was displayed if a message was received. One buffer was included that contained the list of nodes to with which to communicate, and only one type of packet was sent by the nodes.

There was much time spent in writing the code that performs the sensor node and base station functions. The program needed to deal with creating and maintaining the arrival time buffer for each node that is considered a neighbor. The complexity of this section was what made this part of developing the code most difficult. Sensor nodes also needed to respond to packets from the base station appropriately, namely the request for the packet transmission times, the validation of a node, and a broadcast identifying an

intruder. The base station mainly responds to the packets it received. If an ALERT message is received, the base station updates the alert buffer and sends a message to the suspected node requesting its packet transmission times. When the suspected node responds, the base station removes the entry from the alert buffer and checks the contents of the packet to verify whether the node is compromised or not. If it is compromised, a broadcast message is sent. If it is not compromised, the base station sends a message to the node that sent the ALERT giving it the node id of the validated node.

4.2 Example Simulation Run

This section provides a step by step example of what exactly was done when the simulations were run. To begin with, a cygwin window shown below needed to be opened. The 'export dbg = usr3' command allows for the debug messages to be viewed. The third line executed allowed for the TinyViz program to simulate 20 nodes and write all the debug messages into the desired file, which is 'logtest.txt' in the window shown below.

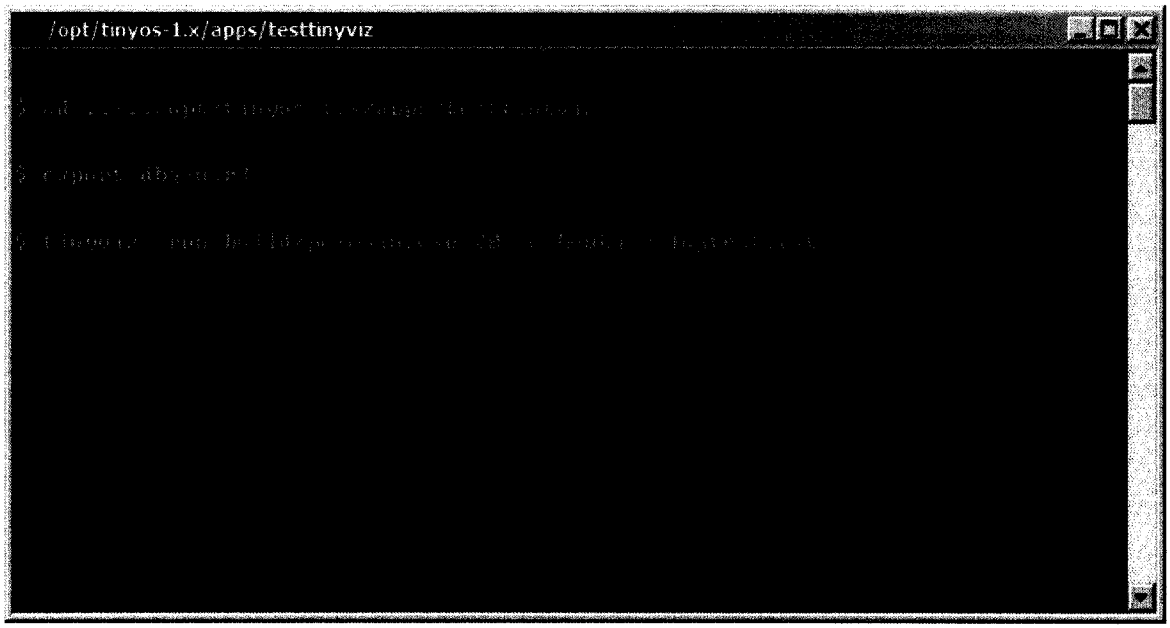

Figure 5. Cygwin window.

The window below shows the TinyViz GUI with the 20 nodes randomly dispersed. The 'Radio model' and 'Debug messages' options were selected from the 'Plugins' menu that is displayed.

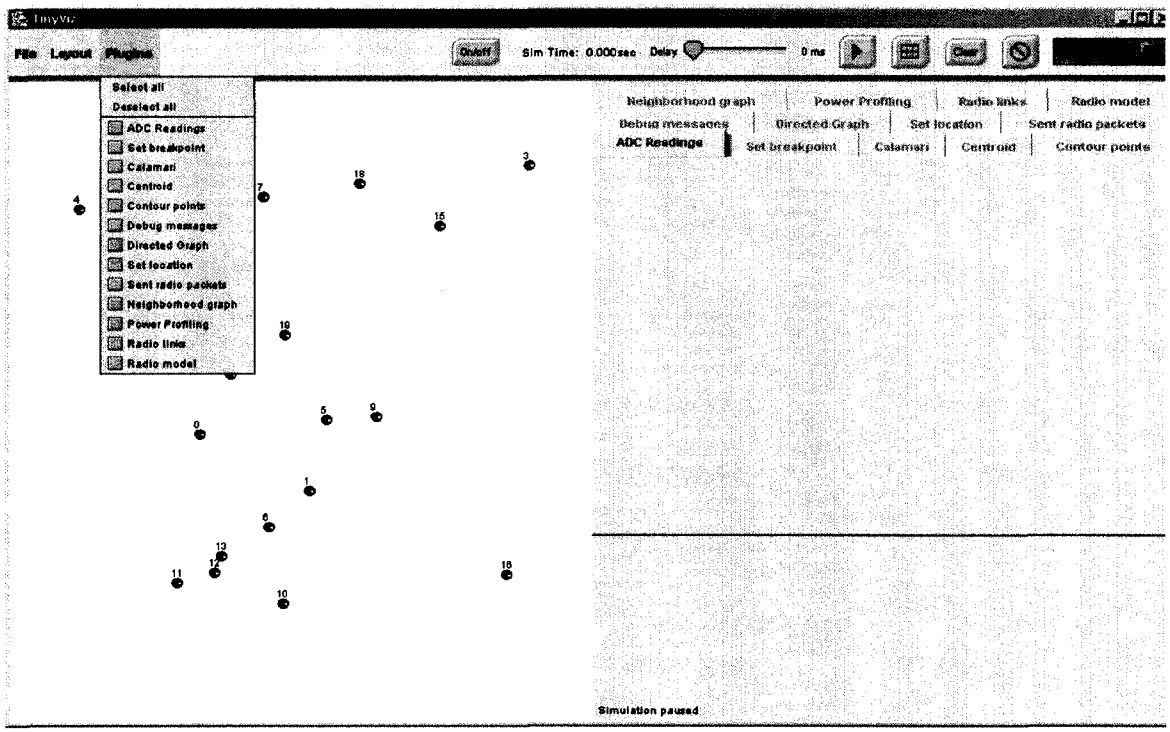

Figure 6. TinyViz running with 20 nodes.

Selecting the 'Radio model' option opens up the tab shown on the right side of the screen. Here, the 'Fixed radius (1000.0)' option was chosen to allow all nodes to communicate with each other.

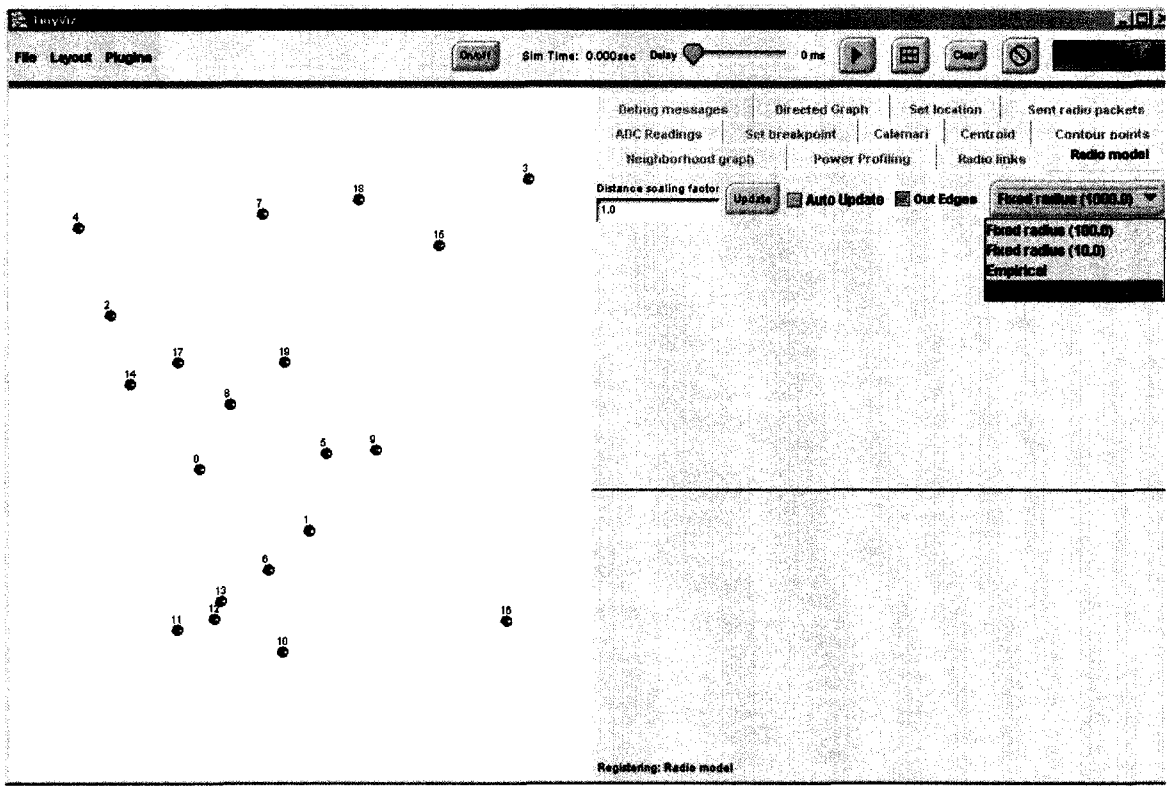

Figure 7. Fixed radius option of Radio model plugin.

Next, the 'Debug messages' option was selected from the 'Plugins' window. The 'Show Radio Messages' option was turned off since it was not needed, and the 'Show Debug Messages' was left on.

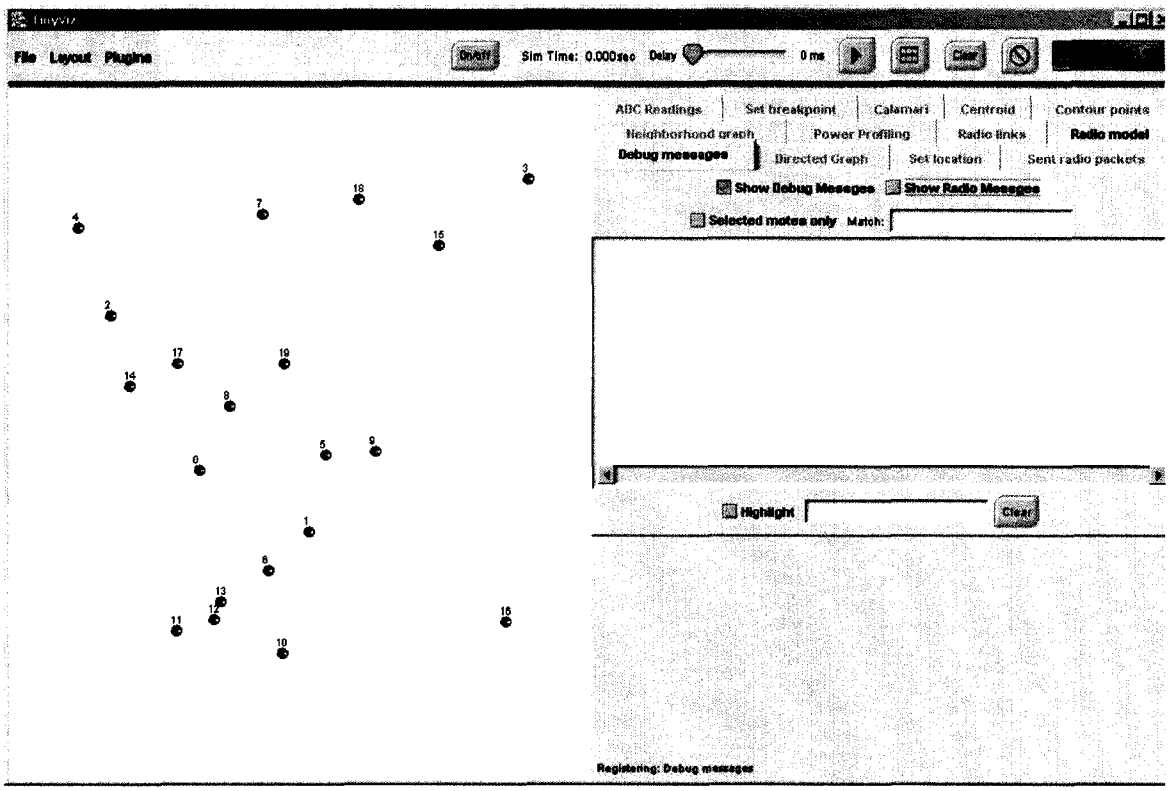

Figure 8. Debug messages plugin.

The green play button at the top of the screen is then pressed to run the simulations. The following window shows the debug messages that are displayed in the TinyViz screen and printed to 'logtest.txt' once this is done.

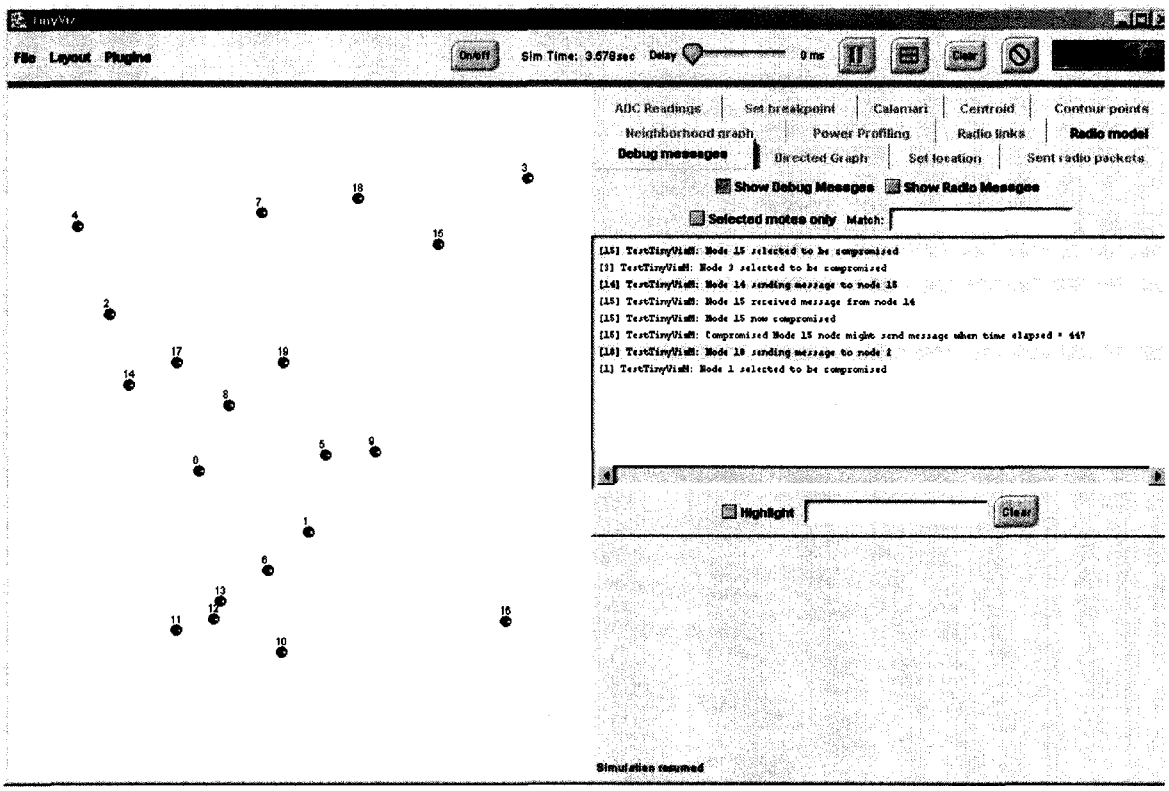

Figure 9. Debug messages displayed in screen.

4.3 Simulator Drawbacks and Configuration

To begin the simulation process, there were several parameters that were altered for numerous runs to determine what kind of effect they had. Increasing the number of packets to obtain while computing the threshold values did not produce a noticeable difference in the time it took to detect a compromised node or the number of times a valid node was wrongly accused of being an intruder. There was a very noticeable difference, though, in the time required for the initialization phase. Therefore, a standard of 10 packets was assigned for all simulation runs regarding the number of packets needed when determining the high and low values of acceptable packet arrival time. Since the event-driven property of the network keeps the packet transmission times of the nodes consistent, the value was kept at 5 throughout simulations.

During these simulations, each sensor node is to send a packet every 3 seconds. The sensor nodes are turned on at different times, meaning the time elapsed since the simulator started can be different from another node. The main reason for this is due to the fact that if all the nodes are turned on at the same time, considering that they all communicate on the same frequency, there would be constant collisions in the network. The packets would all be jammed and communication in the network would drop significantly. Also, in real applications of a wireless sensor network, the sensor nodes would need to be turned on manually and thus would not all have the same time.

One difficulty found in these simulations was that TOSSIM runs the same code on all the nodes. When dealing with actual sensors in a real sensor network, the program that is needed by that particular node is downloaded into its memory. If unique node identification numbers are required for a sensor node, they are easily established during this process. When working with Crossbow MTS310CA sensor and MPR400CB processor radio board, which together made a sensor node, this task was accomplished by simply typing the node number in the command line used to download the code. Any program needed for the sensor nodes should not contain the code required for the base station to perform its tasks or the code that makes the compromised node perform its malicious activities. However, when dealing with TOSSIM, the only noticeable method to have separate code for the valid sensors, the compromised nodes, and the base station was to have one program for all three while having the node id checked when certain function calls were made. This means there is additional time needed to check for which type of node is being simulated as well as additional storage space needed for the code.

It was also speculated that there could be problems introduced into the network when one node was waiting on a response from another node. For example, if the base

station was waiting for a suspected node to send its transmission time buffer, it might be held up for an unnecessary or even an indefinite amount of time if the packet is lost for reasons such as congestion in the network. This led to the elimination of any planned wait functions. In order to compensate, the alert buffer was created for the base station to get the transmission times buffer from suspected nodes in the situation where the request is lost before reaching its destination.

Another hurdle came about since there was no method found to differentiate the nodes in terms of hardware. It was decided that the sensor nodes of the network are to have the same hardware and protocol stack running on them. Since a compromised node in this algorithm is defined as a node with code that performs malicious activities, the same goes for these nodes. However, base stations are generally expected to have more resources available to them. In this algorithm, the base station needed to be able to reach every all nodes of the network. Since there was no method seen in TOSSIM to accomplish this, every node in the network was set to be able to communicate with every other node. Initially, a node would consider another node a neighbor when it received a packet from said node. When the nodes are spread out over a given area, neighboring nodes are normally those that are closest in communication range. To deal with the modification in the algorithm that every node can communicate with every other node and the signal strength is the same, the code had to be altered such that each node was assigned a set of nodes to be its neighbors.

Before the simulations to test the efficiency of the designed algorithm were run, the time needed for the initialization phase needed to be determined. For each network size used, simulations were run to see how long it took for each node of the network to gather the threshold values for each of its neighbors. A compromised node would be

introduced into the network after the initialization phase was said to have finished. The larger the network size, the more time allotted for the initialization phase. The 50 node simulations were given 45 minutes for the initialization phase. It is important to note that the simulations were observed to occur in real time.

V. SIMULATIONS & ANALYSIS

The next simulations that were run involved increasing the number of nodes in the network while keeping the percentage of compromised nodes introduced into the network the same. Three simulations were run for each network size. Figures 10 and 11 below show the time associated with detecting the compromised nodes. Figure 10 displays the average time to detect each compromised node in the network. Once a node becomes compromised, the time it takes before the base station alerts the rest of the nodes to its presence is recorded and averaged for each set of simulation runs. Figure 11 displays the average time it takes to locate all compromised nodes once they are introduced into the network. During these simulations, all compromised nodes are introduced around the same time. The average time elapsed for each simulation from when the first node is compromised to when the last compromised node is detected is shown here. A 95% confidence interval for the sample mean was calculated, and a difference was observed for each network size. The largest interval was seen for the 50 node networks with respect to the time it took to detect each compromised node, which was 644.07 +/- 151.69, as well as for all compromised nodes, which was 1127 +/- 229.33. Increasing the number of simulations for each network size from 3 to 30 would produce a smaller interval around the mean. This goes for all of the following figures included.

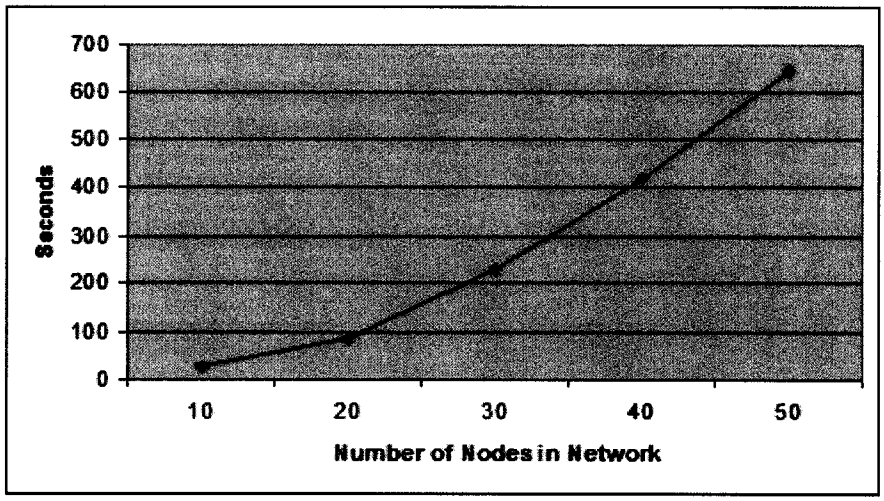

Figure 10. Average Time to Detect Each Compromised Node.

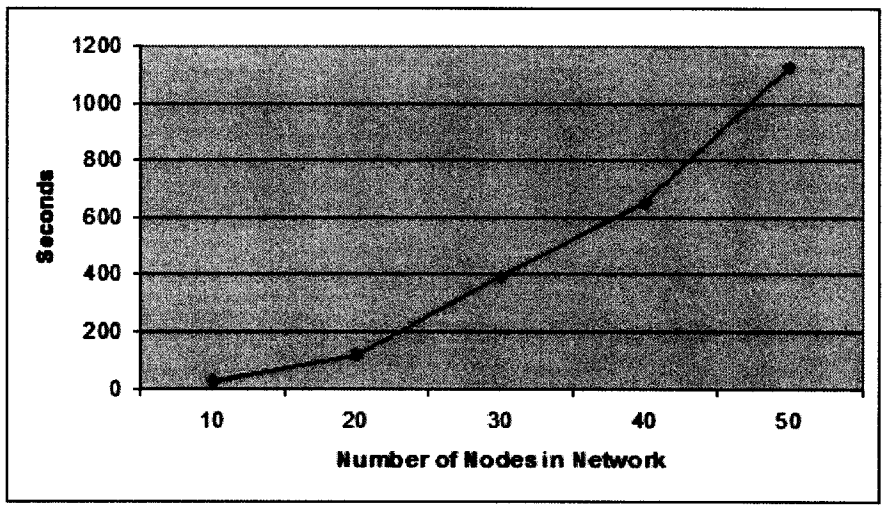

Figure 11. Average Time to Detect All Compromised Nodes

What needs to be considered when looking at these graphs is that 10 % of the number of nodes in the network was compromised. This means that the 10 node network size will have 1 compromised node while the 50 node network size will have 5. Since the base station needs to service all ALERT messages, the time to needed to identify the additional nodes is higher in the larger sized networks. This attributed to the larger

number of packets propagating through the network which consequently leads to a higher number of dropped packets. The alert messages buffer kept by the base station aided in reducing the number of ALERT messages ignored. A base station with better computational resources would have helped to decrease the amount of time necessary to detect the compromised nodes. Also to keep in mind is that there is only one base station for every network size. It is expected that it would take longer to detect the compromised nodes in a larger wireless sensor network.

Figure 12 shows the average number of compromised packets sent by an intruder in the network. The largest 95% confidence interval calculated was the 10 node network size with an interval of 1.67 +/- 1.31.The numbers on the left indicate the average number of packets that are sent by a compromised node that is a functioning member of the network. This graph indicates that the algorithm designed efficiently stops the selective forwarding attacks before each compromised node can pose a significant problem in the network. Once a compromised node exhibits suspicious behavior that is picked up by the other sensor nodes, it is stopped quickly when considering the number of packets it is able to inject into the network. The slight increase in the number of compromised packets as the network size increases is due to the congestion and dropped packets that occur due to the higher amount of traffic. It was often noted that the base station would not be able to respond to multiple ALERT message coming in at the same time. While this is acceptable to some degree since it is a normal occurrence in wireless sensor networks, the extent to which it happened could probably have been reduced if the base station had more resources available to it.

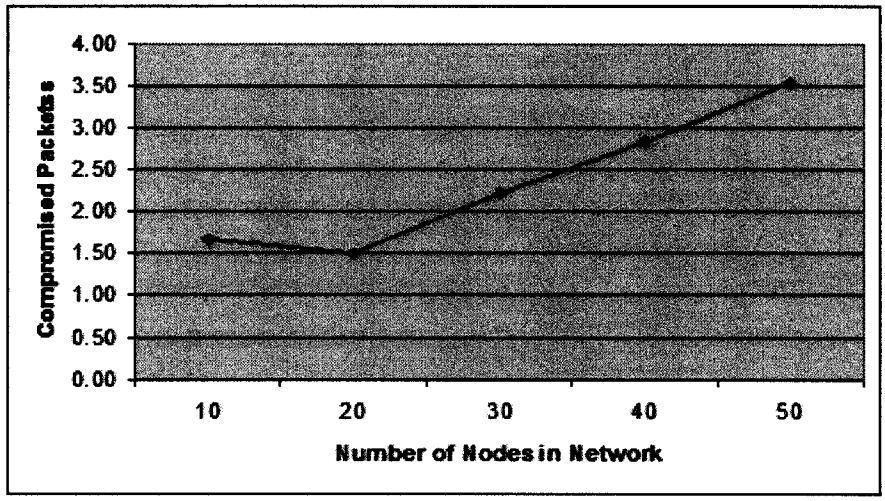

Figure 12. Average Number of Compromised Packets Sent per Compromised Node.

Figure 13 depicts the average number of false positives that occur during the simulation runs for each network size. The largest 95% confidence interval calculated was the 40 node network size with an interval of 216 +/- 31.1. False positives occur when a valid node of the network is labeled as a compromised node. This figure represents the number of times the base station receives an ALERT message from one of the sensor nodes containing an uncompromised node id and proceeds to determine that the node is in fact a valid node. Obviously, as more packets are being sent in the larger network sizes, more ALERT messages would be sent. The base station provides an important service in distinguishing the compromised nodes from the valid nodes. All of the false positives are prevented from occurring in this algorithm. Based on my knowledge, I believe that the number of false positives occurring in other anomaly-based intrusion detection systems is non-zero.

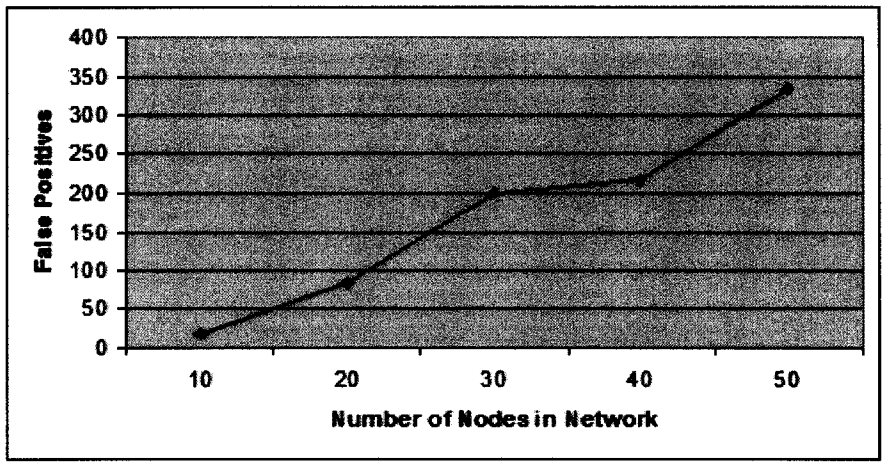

Figure 13. Average Number of False Positives Prevented.

In the next set of simulations, the network size was kept at a constant value of 20 while the percentage of compromised nodes was increased. The time required to detect each compromised may slightly increase as the number of compromised nodes in the network increases as seen in Figure 14. This is due in part to the selective forwarding attack being performed by the compromised nodes. Since the nodes do not detect the anomalies in packet arrival times until a packet is actually sent, it takes more time to detect them when there are more performing the same attack which is seen in Figure 15. The largest 95% confidence interval calculated for the time it took to detect each compromised node was observed with 2 compromised nodes with an interval of 83.2 +/- 69.8. The largest 95% confidence interval calculated for the time it took to detect all compromised node was again seen with 2 compromised nodes but with an interval of 120 +/- 107. The slight wave noticed in Figure 14 is due to the fact that not all the compromised nodes were detected during the some of the simulation run when there were 8 and 10 compromised nodes. The importance of these figures is that as the number of

compromised nodes increases, the nodes and the base station are still able to work together to discover the intruders.

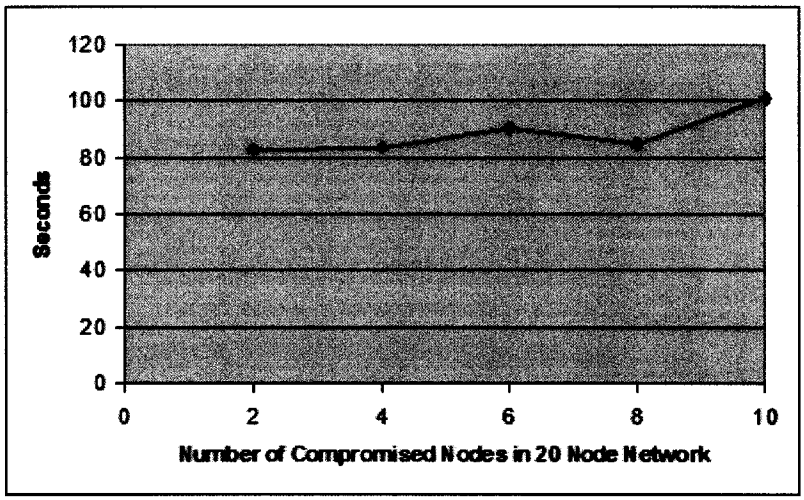

Figure 14. Average Time to Detect a Compromised Node in a 20 Node Network.

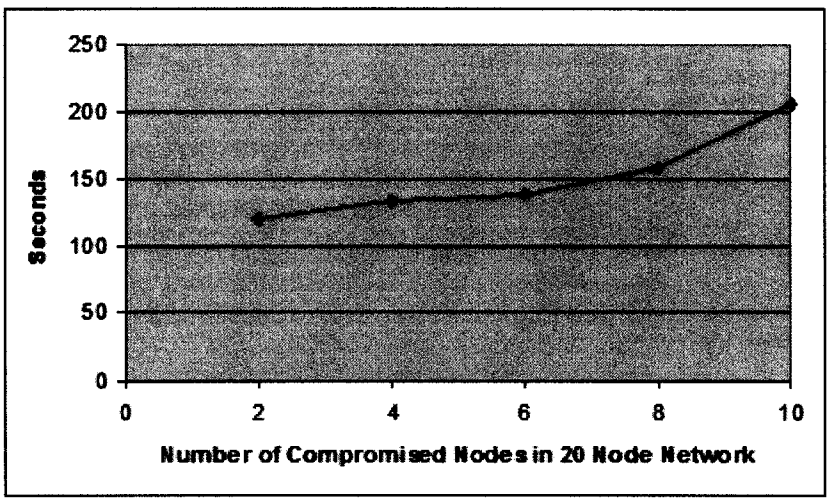

Figure 15. Average Time to Detect All Compromised Nodes in a 20 Node Network.

Figure 16 shows the effect of the compromised nodes in a 20 node network regarding the number of packets that are sent while the node is compromised. The largest

95% confidence interval calculated was seen to occur with 10 compromised nodes that had an interval of 2.63 +/- 1.76. Examining the results shown in Figure 14 and 16 together indicates that the malicious nodes are stopped quickly before they have much time to wreak havoc in the network. The curve in Figure 16 basically implies that the average number of compromised packets being sent is the same for all nodes since this number fluctuates slightly around 2 packets per compromised nodes, regardless of the number of compromised nodes present in the network.

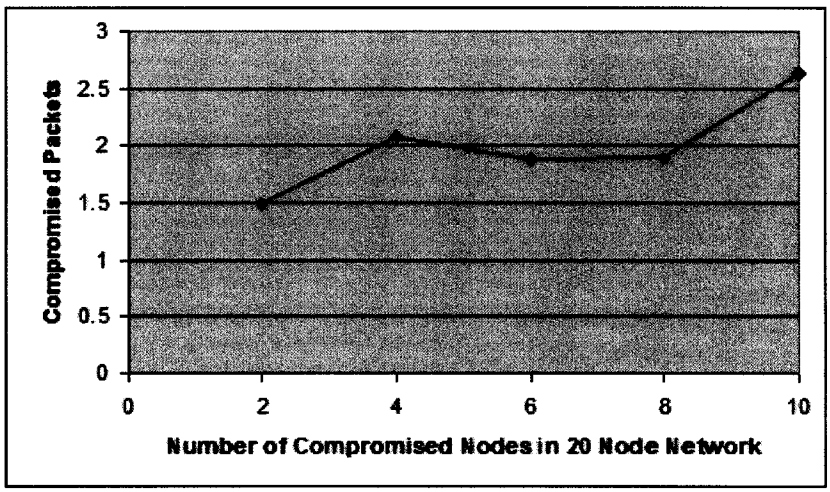

Figure 16. Average Number of Compromised Packets Sent per Compromised Node in a 20 Node Network.

Figure 17 shown below represents the number of false positives that are prevented from occurring within the network. The largest 95% confidence interval calculated for the these numbers was with 10 compromised nodes with an interval of 58.3 +/- 11.3. What is interesting to note here is that the number of false positives decreases as the network size increases. This figure can be explained by considering the fact the network size is staying the same while the number of compromised nodes performing the selective

forwarding the attack increases. Since more nodes are not sending packets, this means the number of messages sent throughout the network is decreased.

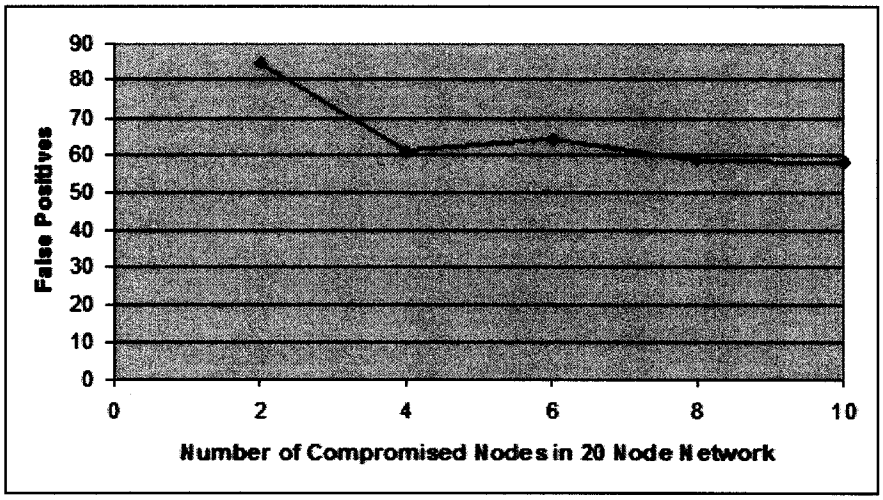

Figure 17. Average Number of False Positives Prevented in a 20 Node Network.

Figure 18 below shows the overhead associated with implementing the designed algorithm in a 20 node network with 2 compromised nodes. The packets represented by the blue shaded region are those sent by the sensors nodes during normal operation. The 95% confidence interval calculated for these packets to be sent is 1382.33 +/- 27.28. The packets in the purple shaded area represent the communication involved with identifying compromised nodes. This including the ALERT messages sent to the base station, the base station requesting packet transmission time buffers, the response the suspected node sends back, and the broadcast messages sent by the base station identifying compromised nodes. The 95% confidence interval calculated for compromised nodes packets is 358.67 +/- 34.79. The packets sent by the base station indicating valid nodes are represented in the yellow shaded area. The 95% confidence interval calculated with respect to these packets being sent is 84.67 +/- 6.82. There is a decrease in the number of packets

associated with normal operation since the sensor nodes are event-driven, and it takes time to construct, process, and receive packets.

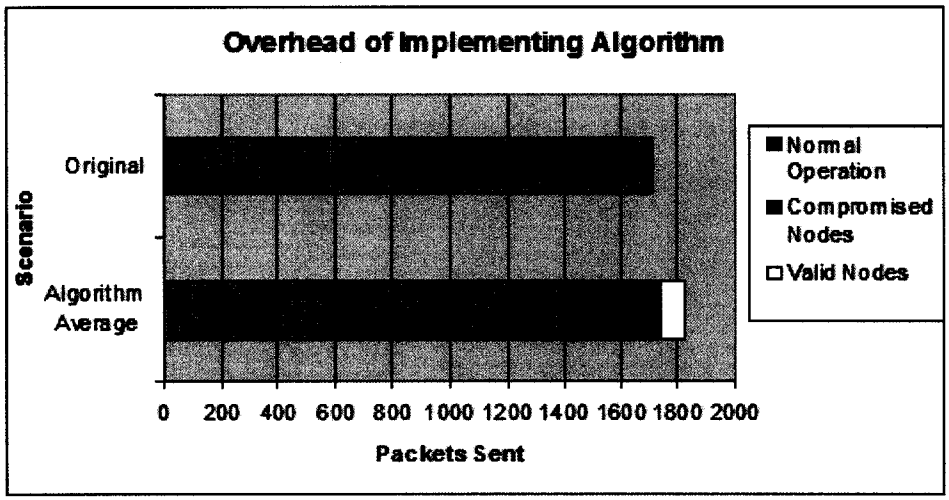

Figure 18. Overhead of Implementing Algorithm in a 20 Node Network.

VI. CONCLUSION AND FUTURE WORK

An algorithm for detecting compromised nodes in wireless sensor networks was designed and verified in this thesis. The anomaly-based intrusion detection technique was applied by utilizing the concept that a valid node of the network will follow an expected pattern of behavior. In other words, deviation from the norm is grounds for suspicion which allows a node participating in the network to identify intruders. Each sensor node was required to maintain a buffer that contained threshold values of packet arrival times for each node it considered a neighbor. If a neighbor's packet fell outside of this desired range, the node would send an ALERT message to the base station informing it of the presence of a possible intruder.

Other implementations of this technique mandate that a node must acquire a preset number of messages before labeling a node as compromised. Before this point, the node in question is just a suspect. This is done to reach some sort of a consensus in the network regarding the identity of intruders so that the number of false positives is decreased. Once a malicious node has been deemed compromised, this information is then propagated to the rest of the network. In this thesis, the base station, a trusted entity in the network, performs the function of these other nodes, with good accuracy. By having the base station investigate all claims of compromised node through means of checking the packet transmission times buffer, there were no occurrences of misidentifying an uncompromised node. This is very important since false positives are prevalent in many anomaly-based intrusion detection systems. The information was also spread quickly since multiple nodes did not have reach the same conclusion before the decision was made.

This thesis uses the event-driven characteristics of sensor networks to allow the base station to determine whether a node is compromised. Since the transmission of packets being sent is centered around a timer as well, each sensor node was instructed to keep a buffer pertaining to these packet transmission times. When the base station is alerted to claims of abnormal behavior, it verifies them by checking the expected difference in packet transmission times of the suspected node versus the actual difference.

While the simulator chosen was not the most ideal, it did enable the designed protocol to be tested for efficiency. The purpose of this thesis was to accurately detect the presence of compromised nodes running the selective forwarding attack in a timely manner. This was shown in the figures depicting the results of the simulations. The compromised nodes were identified by the other nodes of the network and verified by the base station in a reasonable amount of time. This was seen in larger network sizes with the same percentage of compromised nodes as well as in the same network size with different percentages of compromised nodes. The number of false positives that were prevented provides another indication of the importance of the base station.

Consequently, the base station itself was where the most improvement could be made. The TOSSIM simulator utilized in this thesis had many drawbacks that needed to be overcome to test the efficiency and accuracy of the designed algorithm. If there was a method provided to distinguish between the different types of nodes, mainly the base station and sensor nodes, this would have given a more realistic idea of the capability of this algorithm where the base station is concerned.

REFERENCES

[1] Yee Wei Law and Paul J.M. Havinga, "How to Secure a Wireless Sensor Network," *Proceedings of the 2005 International Conference on Intelligent Sensors, Sensor Networks and Information Processing Conference*, Dec. 2005, pp. 89 – 95.

[2] Adrian Perrig, John Stankovic, and David Wagner, "Security in Wireless Sensor Networks," *Communications of the ACM*, vol. 47, no. 6, June 2004, pp. 53 – 57.

[3] Elaine Shi and Adrian Perrig, "Designing Secure Sensor Networks," *IEEE Wireless Communications*, vol. 11, no. 6, Dec. 2004, pp. 38 – 43.

[4] Germano Guimaraes, Eduardo Souto, Djamel Sadok, and Judith Kelner, "Evaluation of Security Mechanisms in Wireless Sensor Networks," *Proceedings of the 2005 Systems Communications*, 2005, pp. 428 – 433.

[5] Harald Vogt, Matthias Ringwald, and Mario Strasser, "Intrusion Detection and Failure Recovery in Sensor Nodes," available at http://www.vs.inf.ethz.ch/res/papers/vogt05recovery.pdf

[6] Carl Hartung, James Balasalle, and Richard Han, "Node Compromise in Sensor Networks: The Need for Secure Systems," Technical Report CU-CS-990-05, Dept of Comp Science, University of Colorado at Boulder, Jan 2005, available at http://www.cs.colorado.edu/department/publications/reports/docs/CU-CS-990-05.pdf

[7] Amitabh Mishra, Ketan Nadkarni, and Animesh Patcha, "Intrusion Detection in Wireless Ad Hoc Networks," *IEEE Wireless Communications*, vol. 11, no. 1, Feb. 2004, pp. 48 – 60.

[8] Wenliang Du, Lei Fang, and Peng Ning, "LAD: Localization Anomaly Detection for Wireless Sensor Networks," Journal of Parallel and Distributed Computing, vol. 66 , issue 7, July 2006, pp. 874 – 886.

[9] Donggang Liu, Peng Ning, Wenliang Du, "Detecting Malicious Beacon Nodes for Secure Location Discovery in Wireless Sensor Networks," available at http://discovery.csc.ncsu.edu/pubs/icdcs05.pdf

[10] Ilker Onat, Iand Ali Miri, "An Intrusion Detection System for Wireless Sensor Networks," *IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, WiMob'2005*, vol. 3, 2005, pp. 253 – 259.

[11] Arvind Seshadri, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla, "SWATT: softWare-based attestation for embedded devices," *Proceedings of the 2004 IEEE Symposium on Security and* Privacy, May 9-12, 2004, pp. 272-282.

[12] Yoshinori Okazaki, Izuru Sato, and Shigeki Goto, "A New Intrusion Detection Method based on Process Profiling," *Proceedings of the 2002 Symposium on Applications and the Internet 2002 (SAINT 2002)*, Jan. 28 - Feb. 1, 2002, pp. 82 – 90.

[13] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: A survey," *Computer Networks*, vol. 38, no. 4, Mar. 15, 2002, p 393-422.

[14] Marcos August0 M. Vieira, Claudionor N. Coelho Jr., Di6genes Cecilio da Silva Junior, and Jose M. da Mata, "Survey on Wireless Sensor Network Devices," *Proceedings of the ETFA '03 IEEE Conference on Emerging Technologies and Factory Automation*, vol. 1, Sept.16 – 19, 2003, pp. 537 – 544.

[15] Chris Karlof and David Wagner, "Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures," *Proceedings of the First IEEE 2003 IEEE International Workshop on Sensor Network Protocols and Applications*, May 11, 2003, pp. 113 – 127.

[16] Sasha Slijepcevic, Miodrag Potkonjak, Vlasios Tsiatsis, Scott Zimbeck, and Mani B. Srivastava, "On Communication Security in Wireless Ad-Hoc Sensor Networks," *Proceedings on the Eleventh IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2002*, June 10 – 12, 2002, pp. 139 – 144.

[17] Paul Brutch and Calvin Ko, "Challenges in Intrusion Detection for Wireless Ad-hoc Networks," *Proceedings on the 2003 Symposium on Applications and the Internet Workshops*, Jan. 27 – 31, 2003, pp. 363 – 373.

[18] Chris Karlof, Naveen Sastry, and David Wagner, "TinySec: A link layer security architecture for wireless sensor networks," *Proceedings of the Second International Conference on Embedded Networked Sensor Systems*, 2004, pp. 162-175.

APPENDIX

A. TinyViz code used for running TOSSIM

```
// $Id: TestTinyVizM.nc,v 1.2 2003/10/07 21:45:24 idgay Exp $

/*
 * Copyright (c) 2003
 *        The President and Fellows of Harvard College.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE UNIVERSITY AND CONTRIBUTORS ``AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE UNIVERSITY OR CONTRIBUTORS BE
LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

/* Author: Matt Welsh <mdw@eecs.harvard.edu>
 * Last modified: 3 August 2003
 */

/**
 * The TestTinyViz application simply sends random messages to demonstrate
 * the debugging and visualization features of TinyViz.
 * @author Matt Welsh <mdw@eecs.harvard.edu>
 */
module TestTinyVizM {
  provides {
    interface StdControl;
  }
  uses {
    interface Timer;
    interface Time;
```

```
  interface ReceiveMsg;
  interface SendMsg;
  interface Random;
  interface SysTime;
 }
} implementation {

 enum {
  MAX_NEIGHBORS = 3,
 };

 uint16_t neighbors[MAX_NEIGHBORS];
 TOS_Msg beacon_packet;

 uint16_t num_nodes = 20;
 uint16_t time_initial;
 uint16_t compromised = 0;

 command result_t StdControl.init() {
  int i;
  struct timeval tval;
  gettimeofday (&tval, NULL);
  time_initial = localtime (&tval.tv_sec);
  time_initial = tval.tv_sec;

  if (((TOS_LOCAL_ADDRESS % 2) != 0) && (TOS_LOCAL_ADDRESS != 0))
  {
        compromised = 1;
  }

  if (TOS_LOCAL_ADDRESS != 0)
  {
   for (i = 0; i < MAX_NEIGHBORS; i++)
   {
    neighbors[i] = ((TOS_LOCAL_ADDRESS + i + 1) % num_nodes);
    if (neighbors[i]  == 0)
     neighbors[i] += MAX_NEIGHBORS - i;
   }
  }
  *((uint16_t *)beacon_packet.data) = TOS_LOCAL_ADDRESS;
  return call Random.init();
 }

 command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, 3000);

 }
 command result_t StdControl.stop() {
  return call Timer.stop();
 }

 uint16_t packet_tr[5];
 uint16_t packet_counter;
 uint16_t trans_time_prev;
 uint16_t trans_time_curr;
 uint8_t compromised_now = 0;
 bool displayed_compromised_msg;
```

```
uint16_t num_compr_packets_sent = 0;
uint16_t num_total_packets_sent = 0;

int alert_messages[50];
int a;
int alert_added;
int alert_sent;

event result_t Timer.fired() {
  uint16_t nbr;
  uint16_t send_rand = -1;
  struct timeval tval;
  uint16_t time_current;

  trans_time_curr += 1;

  gettimeofday (&tval, NULL);
  time_current = tval.tv_sec;

  if (TOS_LOCAL_ADDRESS == 0)
  {
    int alert_num;
    for (a = 0; a < 50; a++)
    {
      alert_num = (alert_sent + a) % 50;
      if (alert_messages[alert_num] > 0)
      {
        struct SecMsg *pack;
        pack = (struct SecMsg *)beacon_packet.data;
        pack->messtype = 4;
        pack->sourceMoteID = TOS_LOCAL_ADDRESS;
        pack->resentrequest = 1;
        call SendMsg.send(alert_messages[alert_num], sizeof(struct SecMsg), &beacon_packet);
        dbg(DBG_USR3, "TestTinyVizM: Request for packet buffer from alert_messages[%d] = %d\n",
alert_num, alert_messages[alert_num]);
        break;
      }
    }
    if ((time_current - time_initial) > 270)
    {
      dbg(DBG_USR3, "TestTinyVizM: time_current - time_initial = %d\n", time_current - time_initial);
    }
    return SUCCESS;
  }

  if (compromised)
  {
    if ((time_current - time_initial) > 360)
    {
      compromised_now = 1;
      if (displayed_compromised_msg == FALSE)
      {
        dbg(DBG_USR3, "TestTinyVizM: Node %d now compromised \n", TOS_LOCAL_ADDRESS);
        displayed_compromised_msg = TRUE;
      }
      dbg(DBG_USR3, "TestTinyVizM: Compromised Node %d node might send message when time
elapsed = %d\n", TOS_LOCAL_ADDRESS, time_current - time_initial);
```

```
    }
   }

   if ((compromised_now != 1) || ((compromised_now == 1) && ((call Random.rand() % 7) == 0)))
   {
    nbr = call Random.rand() % MAX_NEIGHBORS;

    if (neighbors[nbr] != 0xffff)
    {
     struct SecMsg *pack;

     pack = (struct SecMsg *)beacon_packet.data;
     pack->messtype = 2;
     pack->sourceMoteID = TOS_LOCAL_ADDRESS;

     call SendMsg.send(neighbors[nbr], sizeof(struct SecMsg), &beacon_packet);
     dbg(DBG_USR3, "TestTinyVizM: Node %d sending message to node %d\n",
TOS_LOCAL_ADDRESS, neighbors[nbr]);

     if (compromised)
      num_total_packets_sent += 1;
     if (compromised_now)
     {
      pack->compromisedpacket = 1;
      num_compr_packets_sent += 1;
      dbg(DBG_USR3, "TestTinyVizM: Compromised Node %d Sending packet to Node %d at time =
%d\n", TOS_LOCAL_ADDRESS, neighbors[nbr], time_current);
      dbg(DBG_USR3, "TestTinyVizM: Compromised Node %d has sent %d total packets and %d
compromised packets\n", TOS_LOCAL_ADDRESS, num_total_packets_sent, num_compr_packets_sent);
     }

     packet_tr[packet_counter] = trans_time_curr - trans_time_prev;
     trans_time_prev = trans_time_curr;
     packet_counter = (packet_counter + 1) % 5;
    }
    else
    {
     struct SecMsg *pack;
     pack = (struct SecMsg *)beacon_packet.data;
     pack->messtype = 3;

     pack->sourceMoteID = TOS_LOCAL_ADDRESS;
     call SendMsg.send(TOS_BCAST_ADDR, sizeof(uint16_t), &beacon_packet);
     dbg(DBG_USR3, "TestTinyVizM: Node %d sent Broadcast\n", TOS_LOCAL_ADDRESS);

     packet_tr[packet_counter] = trans_time_curr - trans_time_prev;
     trans_time_prev = trans_time_curr;
     packet_counter = (packet_counter + 1) %5;
    }
   }
   return SUCCESS;
  }

  event result_t SendMsg.sendDone(TOS_MsgPtr msg, bool success) {
   dbg(DBG_USR1, "TestTinyVizM: Done sending, success=%d\n", success);
   return SUCCESS;
  }
```

```
uint16_t packet_arrival_time;
uint16_t packet_arrival_time_difference;

struct timeval tv;

float num_buffer_pkts = 10.0;

typedef struct
{
 int neighbor_id;
 uint16_t last_pkt_arrival_time;
 int total_pkts_added;
 uint16_t high_value;
 int high_packet;
 uint16_t low_value;
 int low_packet;
 int questioned_value;
} neighbor_buffer[MAX_NEIGHBORS];
neighbor_buffer arrival_times;

int num_neighbors = 0;
int m = 0;
uint16_t compromised_nodes[MAX_NEIGHBORS * 2];
int j;
bool nodeaddr_found;
uint16_t compr_pack_found = 0;

event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr recv_packet) {
 int n;
 int p;
 uint16_t nodeaddr;
 struct SecMsg *pack;
 pack = (struct SecMsg *)recv_packet->data;

 nodeaddr = pack->sourceMoteID;

 for (p = 0; p < m; p++)
 {
  if (compromised_nodes[p] == nodeaddr)
  {
   dbg(DBG_USR3, "TestTinyVizM: Node %d is ignoring message rceived from COMPROMISED
NODE %d\n", TOS_LOCAL_ADDRESS, nodeaddr);
   return recv_packet;
  }
 }

 if (TOS_LOCAL_ADDRESS == 0)
 {
  if (pack->messtype == 1)
  {
   bool alertfound;
   int alert_num;

   dbg(DBG_USR3, "TestTinyVizM: Received Alert compromised node - %d message from %d\n",
pack->messdetails[0], pack->sourceMoteID);
   for (a = 0; a < 50; a++)
```

```
      {
       if (alert_messages[a] == pack->messdetails[0])
        alertfound = TRUE;
      }
      if (alertfound == FALSE)
      {
       for (a = 0; a < 50; a++)
       {
        alert_num = (alert_added + a) % 50;
        if (alert_messages[alert_num] == 0)
        {
         alert_messages[alert_num] = pack->messdetails[0];
         alert_added += 1;
         dbg(DBG_USR3, "TestTinyVizM: alert_messages[%d] = %d\n", alert_num,
alert_messages[alert_num]);
         break;
        }
       }
      }
      pack->messtype = 4;
      pack->sourceMoteID = TOS_LOCAL_ADDRESS;
      call SendMsg.send(pack->messdetails[0], sizeof(struct SecMsg), recv_packet);
      //dbg(DBG_USR3, "TestTinyVizM: Request for packet buffer from %d\n", pack->messdetails[0]);
     }
     if (pack->messtype == 5)
     {
      int k;
      int is_compromised = 0;

      dbg(DBG_USR3, "TestTinyVizM: Received packet buffer from %d \n", pack->sourceMoteID);

      if (pack->resentrequest == 1)
       alert_sent += 1;

      for (a = 0; a < 50; a++)
      {
       if (alert_messages[a] == pack->sourceMoteID)
       {
        dbg(DBG_USR3, "TestTinyVizM: Removed alert_messages[%d] = %d\n", a, alert_messages[a]);
        alert_messages[a] = 0;
        break;
       }
      }
      for ( k = 2 ; k < 6; k++)
      {
       if (pack->compromisedpacket == 1)
       {
        dbg(DBG_USR3, "TestTinyVizM: Received packet buffer [%d] = %d \n", k -1, pack-
>messdetails[k-1]);
       }
       if (pack->messdetails[k-1] != pack->messdetails[k])
       {
        compromised_nodes[m] = nodeaddr;
        m = m + 1;
        dbg(DBG_USR3, "TestTinyVizM: Received packet buffer [%d] = %d \n", k , pack-
>messdetails[k]);
```

```
        dbg(DBG_USR3, "TestTinyVizM: Base Station has compromised_nodes[%d] = %d\n", m,
compromised_nodes[m]);
        pack->messtype = 6;
        pack->messdetails[0] = pack->sourceMoteID;
        pack->sourceMoteID = TOS_LOCAL_ADDRESS;
        dbg(DBG_USR3,      "**********************************************      Sending
broadcast message to all that %d IS COMPROMISED ********************\n", pack->messdetails[0]);
        dbg(DBG_USR3, "number of false alarms is %d\n", compr_pack_found);
        call SendMsg.send(TOS_BCAST_ADDR, sizeof(struct SecMsg), recv_packet);
        is_compromised = 1;
        break;
       }
      }
      if (is_compromised == 0)
      {
       dbg(DBG_USR3, "%d is Not Compromised\n", pack->sourceMoteID);
       pack->messtype = 7;
       pack->messdetails[0] = pack->sourceMoteID;
       call SendMsg.send(pack->origsourcenode, sizeof(struct SecMsg), recv_packet);
       compr_pack_found += 1;
      }
     }
    }
    else
    {
     if (pack->messtype == 4)
     {
      int k;
      pack->messtype = 5;
      pack->sourceMoteID = TOS_LOCAL_ADDRESS;
      pack->messdetails[0]=  0; // CONTAINS FUNCTION VALUE
      for ( k = 0 ; k < 6; k++)
       pack->messdetails[k+1] = packet_tr[k];

      call SendMsg.send(0x0, sizeof(struct SecMsg), recv_packet);
      //dbg(DBG_USR3, "TestTinyVizM: pack->origsourcenode = %d \n", pack->origsourcenode);
      dbg(DBG_USR3, "TestTinyVizM: Sending packet buffer to base station from %d \n",
TOS_LOCAL_ADDRESS);
      return recv_packet;
     }
     if (pack->messtype == 6)
     {
      dbg(DBG_USR3, "TestTinyVizM: %d Received broadcast compromised node notification for %d\n",
TOS_LOCAL_ADDRESS, pack->messdetails[0]);

      for (j = 0; j < MAX_NEIGHBORS; j++)
          {
       if (arrival_times[j].neighbor_id == pack->messdetails[0])
       {
        arrival_times[j].neighbor_id = 0xffff;
        arrival_times[j].last_pkt_arrival_time = 0xffff;
        arrival_times[j].total_pkts_added = 0xffff;
        arrival_times[j].high_value = 0xffff;
        arrival_times[j].low_value = 0xffff;
        compromised_nodes[m] = pack->messdetails[0];
        num_neighbors -= 1;
```

```
        dbg(DBG_USR3, "TestTinyVizM: Removed arrival_times[%d] from neighbor list since it is now
compromised_nodes[%d] = %d\n", j, m, compromised_nodes[m]);
        m = m + 1;
       }
      }
      return recv_packet;
    }
    if (pack->messtype == 7)
    {
     dbg(DBG_USR3, "TestTinyVizM: %d Received message from base station that %d is not
compromised\n", TOS_LOCAL_ADDRESS, pack->messdetails[0]);

      for (j = 0; j < MAX_NEIGHBORS; j++)
          {
       if (arrival_times[j].neighbor_id == pack->messdetails[0])
       {
        if (arrival_times[j].questioned_value > arrival_times[j].high_value)
        {
         arrival_times[j].high_value = arrival_times[j].questioned_value;
        }
        else
        {
         arrival_times[j].low_value = arrival_times[j].questioned_value;
        }
        arrival_times[j].total_pkts_added = arrival_times[j].total_pkts_added + 1;
       }
      }
      return recv_packet;
    }
    if (nodeaddr == 0)
         return recv_packet;

     dbg(DBG_USR3, "TestTinyVizM: Node %d received message from node %d\n",
TOS_LOCAL_ADDRESS, nodeaddr);

    nodeaddr_found = FALSE;
    gettimeofday (&tv, NULL);
=    packet_arrival_time = tv.tv_sec;

    for (j = 0; j < MAX_NEIGHBORS; j++)
    {
     if (arrival_times[j].neighbor_id == nodeaddr)
     {
      nodeaddr_found = TRUE;
      packet_arrival_time_difference = packet_arrival_time - arrival_times[j].last_pkt_arrival_time;

      if (arrival_times[j].total_pkts_added < num_buffer_pkts)
      {
       if (arrival_times[j].total_pkts_added == 0)
       {
        arrival_times[j].high_value = packet_arrival_time_difference;
        arrival_times[j].high_packet = 0;
        arrival_times[j].low_value = packet_arrival_time_difference;
        arrival_times[j].low_packet = 0;
       }
       else if (packet_arrival_time_difference > arrival_times[j].high_value)
       {
```

```
          arrival_times[j].high_value = packet_arrival_time_difference;
          arrival_times[j].high_packet = arrival_times[j].total_pkts_added + 1;
         }
         else if (packet_arrival_time_difference < arrival_times[j].low_value)
             {
              arrival_times[j].low_value = packet_arrival_time_difference;
              arrival_times[j].low_packet = arrival_times[j].total_pkts_added + 1;
         }

         arrival_times[j].total_pkts_added = arrival_times[j].total_pkts_added + 1;

         if (arrival_times[j].total_pkts_added == num_buffer_pkts)
         {
          if ((arrival_times[j].neighbor_id % 9) == 0)
          {
           dbg(DBG_USR3, "TestTinyVizM: %d Received message from %d so EQUAL packet
numbers\n", TOS_LOCAL_ADDRESS, nodeaddr);
          }
         }
        }
        else
        {
         if (pack->compromisedpacket == 1)
         {
          dbg(DBG_USR3, "TestTinyVizM: %d Received message from %d LIMIT REACHED\n",
TOS_LOCAL_ADDRESS, nodeaddr);
          dbg(DBG_USR3, "TestTinyVizM: packet_arrival_time_difference = %d\n",
packet_arrival_time_difference);
          dbg(DBG_USR3, "TestTinyVizM: high value = %d\n", arrival_times[j].high_value);
          dbg(DBG_USR3, "TestTinyVizM: low value = %d\n", arrival_times[j].low_value);
         }
         if ((packet_arrival_time_difference < arrival_times[j].low_value) ||
         (packet_arrival_time_difference > arrival_times[j].high_value))
         {
          if (pack->compromisedpacket == 1)
          {
           dbg(DBG_USR3, "TestTinyVizM: ***** COMPROMISED NODE %d suspected on
TOS_LOCAL_ADDRESS = %d *****", nodeaddr, TOS_LOCAL_ADDRESS);
          }
          pack->messtype = 1;
          pack->sourceMoteID = TOS_LOCAL_ADDRESS;
          pack->origsourcenode = TOS_LOCAL_ADDRESS;
          pack->messdetails[0] = nodeaddr;

          arrival_times[j].questioned_value = packet_arrival_time_difference;

          dbg(DBG_USR3, "TestTinyVizM: %d Sending Alert message to base station regarding %d\n",
TOS_LOCAL_ADDRESS, nodeaddr);
          call SendMsg.send(0x0000, sizeof(struct SecMsg), recv_packet);
         }
        }
        arrival_times[j].last_pkt_arrival_time = packet_arrival_time;
       }
      }
      if ((nodeaddr_found == FALSE) && (num_neighbors < MAX_NEIGHBORS))
      {
       arrival_times[num_neighbors].neighbor_id = nodeaddr;
```

```
      arrival_times[num_neighbors].last_pkt_arrival_time = packet_arrival_time;
      arrival_times[num_neighbors].total_pkts_added = 0;
      arrival_times[num_neighbors].high_value = 0;
      arrival_times[num_neighbors].low_value = 0;
      num_neighbors += 1;

      for (n = 0; n < MAX_NEIGHBORS; n++) {
       if (neighbors[n] == 0xffff) {
         neighbors[n] = nodeaddr;
         dbg(DBG_USR3, "TestTinyVizM: Node %d now conisders node %d a neighbor\n",
TOS_LOCAL_ADDRESS, nodeaddr);
         break;
       }
      }
     }

   return recv_packet;
  }
}
```

VITA

NAME: Mary Lisa Mathews

DATE OF BIRTH: August 5, 1981

DEGREES:

- *Bachelor of Science* (Computer Engineering), Drexel University, Philadelphia, PA, June 2004.
- *Master of Science* (Computer Engineering), Old Dominion University, Norfolk, VA, May 2007.