Detection of Covert Communications based on Intentionally Corrupted Frame Check Sequences

by

Ali Najafizadeh

A thesis submitted in Partial Fulfillment

of the Requirements for the Degree of

Master of Applied Science

in

the Faculty of Engineering and Applied Science

and the program of Electrical and Computer Engineering

University of Ontario Institute of Technology

Oshawa, Ontario, Canada

(July, 2011)

Abstract

This thesis presents the establishment of a covert-channel in wireless networks in the form of frames with intentionally corrupted Frame Check Sequences (FCSs). Previous works had alluded to the possibility of using this kind of covert-channel as an attack vector. We modify a simulation tool, called Sinalgo, which is used as a test bed for generating hypothetical scenarios for establishing a covert-channel. Single and Multi-Agent systems have been proposed as behaviour-based intrusion detection mechanisms, which utilize statistical information about network traffic. This utilized statistical information is used to detect covert-channel communications. This work highlights the potential impact of having this attack perpetrated in communications equipment with a low chance of being detected, if properly crafted.

Keywords
Wireless networks, covert-channel, hidden-channel, behavioural intrusion detection

To my dear and lovely wife, Sahar, who stays and supports me all days and nights.

To my Parents, who believe in me and support me.

To three people, Laleh, Ladan, and Sepideh who encourage me all the way to finish.

Acknowledgements

This project was supported in part by the Natural Sciences and Engineering Research Council of Canada and by Defence Research & Development Canada.

I would like to thank my supervisor Dr. Ramiro Liscano to let me join his team and gave me all the support for my thesis.

Also I have to thank my co-supervisor Dr. Miguel Vargas Martin for all his valuable comments and energy during my master.

At last I have to thank all my friends specially Dr. Anand Dersingh and John Khalil.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

FCS: Frame Check Sequence

CRC: Cyclic Redundancy Check

SINR: Signal Interference plus Noise Ratio

IDS: Intrusion detection System

VSAAD: Variance-based Single Agent Anomaly Detection

Chapter 1: Introduction

1.1 Motivating Scenario

During the past two decades people have been moving from limited access to information into ultimate access to information. To achieve ultimate access, new technology is demanded. Wired connectivity has been introduced since the beginning of computer networks, but the introduction of wireless networks creates a plethora of new applications and demands. Basically it frees the devices from hard connections such as wires and cables. Unfortunately it introduces new types of problems. Because of the nature of wireless, anyone can receive and access data as long as they are in the range of the signal. So the first rules of thumb would be, if you have the signal you have the data sent in the air. This simple feature makes the wireless system really vulnerable to some attacks. That's why a large number of researchers are interested in the field of wireless security.

1.2 Problem Statement

There are so many attacks aimed at exploiting wireless communications. So the following four items represent the basic but overall steps of any attacks.

- Breaking into a system
- Collecting sensitive data
- Transmitting data back to an attacker
- Erasing all traces of an attack

As you may see, every attack has a start point of breaking into the system. Breaking into any system means passing security defences such as firewalls or intrusion detection systems (IDS). Once the attacker is inside the system, he starts looking for data. Looking for gathering meaningful data can be called collecting data. The next category of attack is how the attacker can transmit data back to herself/himself without being spotted or raising any security alerts. Once the transfer is done, the attacker has to clean up the traces such as log files, IP (Internet Protocol) address caches, etc. to avoid being found.

This research investigates the third item on the attack list, sending data back to the attacker, by hiding communications from the system protection such as Intrusion Detection System (IDS) using purposely corrupted frames.

1.3 Thesis Contribution

The main objectives of this thesis are as follows:

- Creating new plug-in software for a well-known simulator called Sinalgo [1] to facilitate further study of covert-channel attacks.
- Investigating and applying well-known and studied anomaly detection algorithms to see if they are suitable for identifying covert-channel attacks.
- Proposing two new systems, a Single Agent, and a Multi-Agent System to detect covert-channel anomalies based on purposely corrupted FCS.

1.4 Thesis Structure

This thesis is structured as follows:

Chapter 2 covers background and related work focusing on steganography and different types of networked communications such as side-channel and covert-channel. Chapter 3 introduces the Sinalgo [**1**] network simulator and the modifications which are needed in order to support covert channel communications based on purposely corrupted frames. Chapter 4 overviews statistical approaches which intends to find anomalies. Chapter 5 presents experimental results and evaluation of Single Agent and Multi Agent Systems. Chapter 6 talks about the possible implementation of covert channel communications based on purposely corrupted frames on "real" hardware and finally in Chapter 7 the conclusion is given.

Chapter 2: Background and Related Work

2.1 Steganography

Steganography is the art and science of hiding information. It has played an important role in history for spies and militaries for centuries. As an example, in 440 BC, Histiaeus, who shaved the head of his slaves, tattooed messages on their bald heads and once their hair had grown back, sent those men into the enemy lines to deliver the messages. Since that day, steganography has changed enormously. It has been changed from physical into digital to suite and fit in our modern world. Steganography has been adapted into so many technologies. The following list describes some of the adaptations of steganography.

a) Digital Steganography

   Modern steganography entered in our modern world in 1985 with the use of the personal computer being applied to classical steganography problems [**2**]**.** At the beginning adoption was slow, but it has taken off, going by the number of "stego" programs available. There are more than 800 digital steganography applications that have been identified by Steganography Analysis Research Center.

b) Network Steganography

   The general term of network steganography can be classified as information-hiding techniques which may be used to exchange steganograms in telecommunication networks messages such as frames and packets. In 2003, Szczypiorski et al. [**3**] introduced the concept of corrupted CRCs for hidden communication. By using the

digital steganography applications and applying typical steganographic methods which utilize digital data such as images, audio and video files as a cover for hidden information, this combination is very hard to detect which has been described by Szczypiorski.

c) Text Steganography

Although steganography can be applied to different and many types of media such as audio, image, video, and text, text steganography is considered to be one of the most difficult types of steganography because of its relatively low redundancy compared to images and audio files. Text typically uses less memory compared to audio and video, and therefore it can be also used as a simple communication method.

Data compression can also be used and applied as one of the methods for text steganography. Data compression encodes information from one representation to another typically yielding a smaller piece of data. In terms of compression algorithms, Huffman coding [**4**] achieves one of the best compression rates so far known, and its simplicity has helped it gain popularity over the years. Based on Huffman coding, "smaller length code words are assigned to more frequently occurring source symbols and longer length code words are assigned to less frequently occurring source symbols".

d) Printed Steganography

Digital steganography output may be in the form of printed documents. A message, the plaintext, may be first encrypted by traditional means, producing a cipher-text. Then, an

innocuous covert text is modified in some way so as to contain the cipher-text, resulting in the stegotext. For example, the letter size, spacing, typeface, or other characteristic of a covert text can be manipulated to carry the hidden message. Only a recipient who knows the technique used can recover the message and then decrypt it (see e.g. [**5**]). The cipher-text produced by most digital steganography methods, however, is not printable. Traditional digital methods rely on perturbing noise in the channel file to hide the message; as such, the channel file must be transmitted to the recipient with no additional noise from the transmission. Printing introduces too much noise in the cipher-text, generally rendering the message unrecoverable. There are techniques that address this limitation; one notable example is ASCII Art Steganography [**6**].

2.2 Wireless Covert-Channel

Wireless communication is prone to packet errors caused by signal fading, collision, or shadowing effects. In packet-based communications, any corruption to a packet is detected by using a Cyclic Redundancy Check (CRC) on the frame payload and appending information derived from this CRC into a Frame Check Sequence (FCS) field in the frame. Thus the FCS and the CRC serve similar purposes, but in different layers of the communication protocol.

This mechanism can also be used to establish a form of covert channel communication between rogue nodes in a wireless network. These rogue nodes can simply use a different FCS algorithm than the other nodes to communicate with each other. The idea behind this form of covert channel communication is to use frames with intentionally corrupted FCSs as the means for transmitting information between two wireless devices that are within communication range of

each other. This form of covert communication is difficult to detect because it is basically interpreted as noise by the other nodes and corrupted frames will be dropped immediately by these nodes.

An intrinsic property that works in favour of covert-channel communications is that they hide amongst frames with naturally corrupted FCSs, thus making detection harder. In other words, the complete set of frames transmitted in a wireless network is composed of frames with: (a) non-corrupted FCS; (b) naturally corrupted FCS; (c) intentionally corrupted FCS for covert-channel communication; and (d) naturally corrupted FCS of messages that had been intentionally corrupted for covert-channel communication. Thus, the problem of detecting the existence of a covert-channel can be equivalent to telling apart naturally and intentionally corrupted FCSs. The difficulty in detection depends on a number of factors, including the proportion of intentionally corrupted FCSs with respect to naturally corrupted FCSs, and the traffic volumes.

The concept behind our covert communication channel can be illustrated in simple terms by the Alice and Bob analogy used in the context of cryptography. Assume that Alice and Bob are in prison and they plan to communicate with each other. Unfortunately, the only way they can communicate is through Walter, the warden. However, If Walter finds out that Alice and Bob are communicating, he will close their connection. Walter is the network manager and he monitors network traffic. Alice and Bob cannot use cryptography to hide their messages; otherwise Warden would detect there is a communication going on. They have two options; they either use steganography techniques to hide their information in valid network packets or

they hide information in seemingly valid network packets. In our case, they choose the latter by corrupting the FCS to make packets seem naturally corrupted and thus be ignored and discarded by the warden.

The use of hidden covert-channels has been proposed before (see e.g., Szczypiorski [**3**]) but to the best of our knowledge, there is no implementation in wireless devices reported in the literature. The reason is a lack of hardware implementations due to the fact that most devices are shipped with a locked Layer 2 chipsets with another chip. This chip, which is shown in Figure 1, calculates the FCS and passes the new frame to physical layer. This limitation has led researchers to use simulators instead of actual devices.

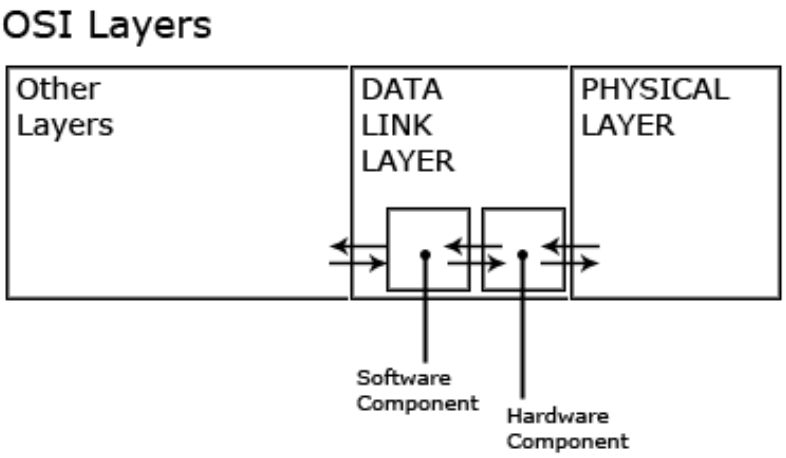

Figure 1: Hardware locked for custom FCS calculation

Odor et al. [**7**] presented an algorithm that could be used for covert-channel communications using an intentionally corrupted FCS in Layer 2 of the protocol stack and performed a simple simulation utilizing Matlab/Simulink. To build on Odor et al.'s work, there was a need to build an actual experimental test-bed as well as the exploration of detection mechanisms in an

appropriate simulator. However because of the way it was designed, it was very challenging to extend his work to multi-node scenarios.

2.3 Related work

As mentioned in the previous section, the focus of this research is on specific types of steganography in the network protocol. Jankowski et al. [**8**] studied a new way of exchanging steganograms data in LAN environments by using the ARP protocol in conjunction with improper Ethernet frame padding to provide localization and identification of the member of a hidden group.

Theodore et al. [**9**] has studied steganography in the seven layers of the OSI network model. In that paper they have pointed out some of the reasons why this model is susceptible to data hiding such as design errors, project deadlines, production shortcuts, etc.

Wolf et al. [**10**] proposed a steganography method for IEEE 802.3 frames which makes it possible to have a hidden bandwidth of up to 45 bytes per frames.

Fisk et al. [**11**] reported on a method that uses the padding of TCP and IP headers in the context of active warden to have 31 bits per packet for steganographic communication. IPv6 has also been studied for steganographic communication. Lucena et al. [**12**] also used padding in IPv6. This technique offers multiple channels with up to 256 bytes per packet for that type of communication.

Two fields of study related to our work are termed covert channel and side-channel. Based on Wagner [**13**] and Murdoch [**14**], there are four different channels for communication. According

to Murdoch, a covert channel can be described as a communication in a computer system where the sender and the receiver plan to leak information over a channel which is not designed for that communication to take place, in violation of a required access control policy. On the other hand, side-channel is a communication over a channel which fails to comply with the security requirements where the sender leaks information inadvertently and the receiver decides whether or not to take advantage and get the information. Thus the biggest difference between a steganographic channel as opposed to a covert or side-channel is that a steganographic channel uses an open channel for communication where both the sender and the receiver plan to prevent observers from detecting that channel. Furthermore, the concept of subliminal channels refers to a covert channel in a cryptographic algorithm which is undetectable. Thus, a subliminal channel is a special case of a steganographic channel, while a steganographic channel can be used to create a covert channel, not all covert channels fall into the category of steganographic channels; a simple example of that would be watermarking. Watermarking is a technique where a message is hidden in a file, such as image, audio or video, in such a way that the source can be verified.

Szczypiorski et al. [**3**] introduced HICCUPS, a steganographic system capable of establishing a side-channel in the data link layer in a wireless LAN. HICCUPS leverages natural fading and interference in wireless signals to mask side-channel messages.

Szczypiorski suggested three features that make a network suitable for HICCUPS. First, the medium has to be shared among all nodes and sender and receiver must have a high probability of intercepting the messages. The second one is having an environment with a

published algorithm for link layer cipher. The last thing is that the environment has to offer some mechanism to validate the integrity of messages. An example would be FCS. If all these three features exist in a system, then three types of hidden channels could be introduced. The first channel would be based on manipulation of the IVs (Initialization Vector) in the link layer encryption. Forging MAC addresses physically in link layer hardware would be the second channel. The third type of channel is based on the data integrity mechanism (e.g., CRC or FCS). These entire three features can be found in IEEE 802.11 with WEP encryption.

Sbrusch et al. [**15**] suggested that almost 25% of wireless transmission is corrupted because of various reasons such as interference and noise in the environment. This high error rate opens a door for sending secret messages.

Szczypiorski also introduced two methods for encoding and transporting covert messages. The "basic mode" utilizes covert messages in network interface MAC address and cipher IVs. In this case, there are only 216 bits in total per frame that can be used to transfer messages. Because of limited transfer rates of side-channels, Szczypiorski suggested to use them to exchange control messages. The "Corrupted frame mode" is introduced to be used for higher bandwidth. In this mode, the entire payload of a frame can be used to transfer the covert messages. Also the FCS, the last four bytes of each frame, can be altered in such a way to notify other HICCUPS members that a covert message is being delivered inside the payload. Szczypiorski estimated that in 802.11b, 44kbits per second and in 802.11g, 216 Kbits per second can be transmitted. Even though HICCUPS offers very high bandwidth, it suffers from a serious drawback: to the

best of our knowledge, there is no commercial wireless card that can actually inject and alter FCSs in frames via existing software and hardware.

Chapter 3: Sinalgo Simulator and New Modifications

The primary objectives of the simulation are:

1. Create a test-bed for investigating covert-channel communications.
2. Analyze statistical properties of traffic in a simple scenario where purposely corrupted frames are injected, and help understand the level to which these frames can be detected when mixed with normal traffic. Here we emphasize that the implementation of a MAC layer was not strictly necessary for these purposes.
3. Leverage our findings under Objective 2 to determine the feasibility of detecting the presence of a covert-channel.

Oder et al. [**7**] provides an outline of the limitations of simulation environments such as NS-2 [**16**], Omnet++ [**17**], Qualnet [**18**], and MATLAB/Simulink [**19**] for use in covert-channel communications. As a result of that review the MATLAB/Simulink simulator was chosen over others, primarily due to the fact that it was one of the few environments where corrupt frames could be received by a node. The nature of Simulink and the way Odor et al. [**7**] designed the framework, the simulation imposed a limit of two nodes which made this research move to a different simulation platform; one that was closer to a network simulator as opposed to a communication channel simulator like Simulink.

Choosing the right simulator, though, is very challenging for the following reasons:

- Some simulators take a significant amount of time and effort to evaluate and understand how they can be programmed; our experience has shown that four to six months is not unusual.
- There are a number of well-known network simulators to choose from; for example, NS-2, Qualnet, Omnet++, and Opnet [**20**]. Most of these are relatively sophisticated platforms because they simulate many layers in the OSI networking stack. Modifying one layer, in particular a lower layer requires very good understanding of its interaction with the other layers.
- Most of the network simulators do not allow the intentional corruption of frames into the communication packets and they certainly do not allow these packets to be sent to other nodes. Typically the simulator will decide for each round which packets are corrupted based on some well-studied channel propagation model and discard the packet before placing it in the receiving nodes' incoming buffer.

Sinalgo is a network simulator which can be modified to allow for crafting invalid FCSs. Sinalgo offers many features and all of those features are nicely wrapped in a simple and well-structured design. Furthermore, Sinalgo is open source and is written in Java, making it fully portable; and allowing for easy modifications and additions to its source code, compared to other simulators such as Qualnet or NS-2. Additionally, Sinalgo comes with a complete and user-friendly set of documentation. Nevertheless, the most important limitation of Sinalgo, for our purposes, is that nodes are not allowed to receive corrupted frames; once a frame becomes corrupted, the frame will not even be sent to its destination. This limitation is bypassed and described in section 3.3.1.

3.1 Overview of Sinalgo

Sinalgo is a java-based simulator which is designed to be simple and effective for wireless nodes. The main interface is designed to be simple as displayed in Figure 2.

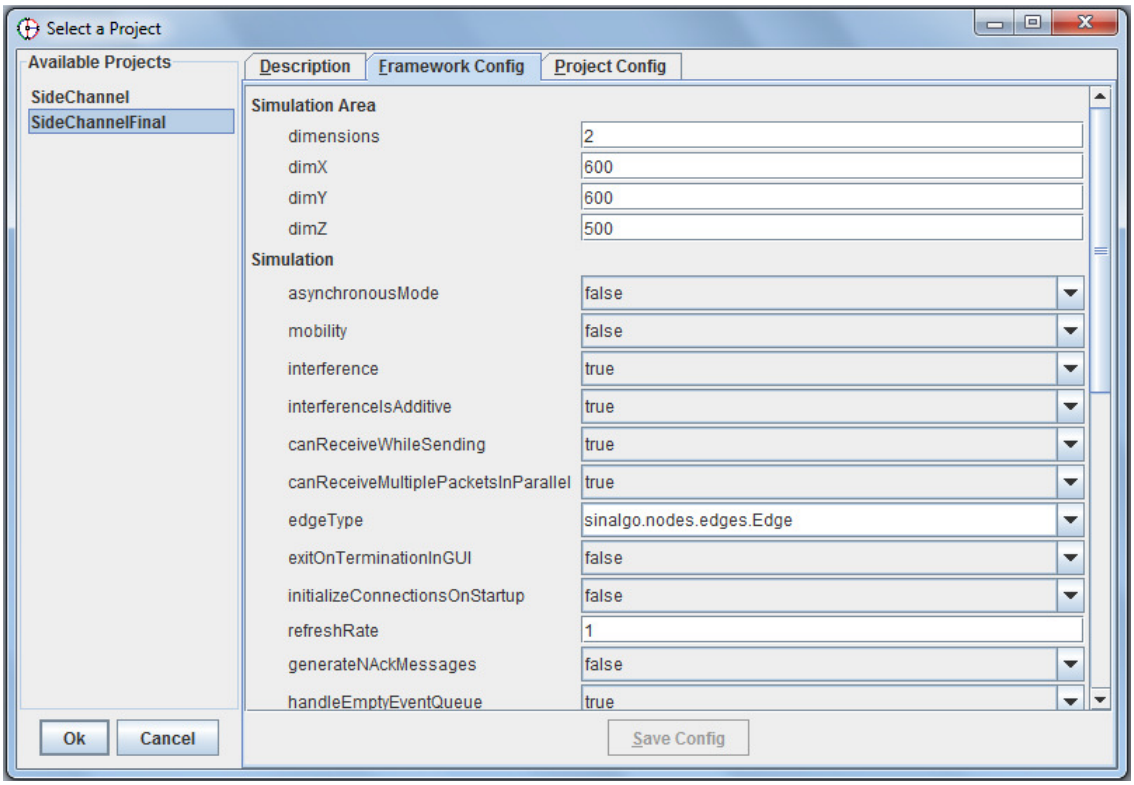

Figure 2: Interface for the Configuration of a Sinalgo simulation

The left side bar shows the current projects. By clicking on each project the right side of the window will change. The right-side window is designed to be a configuration window. This window also populates from an xml file which is included in the project folder of Sinalgo. Sinalgo also provides a mechanism to parse an XML file where it is possible to add new parameters for the simulation. You can also access all the parameters in this XML file during the

simulation and modify it on the fly. It's very handy and flexible because most of the time you just want to change the some parameters and modifying the source code could be a hassle.

In Project Config, all the custom tags which are not default tags are displayed as a raw text. As an example, changing the range of signal for all nodes would be a variable which needed a custom XML field called NodeRange. So from now on, changing the value of that tag will change the range of all nodes which makes it really easy and consistent.

Once the project is selected and configured according to some configurations, it is time to run the simulation. In order to run the simulation, select the Project from the left side bar and click the OK button, which opens another window as shown in Figure 3.

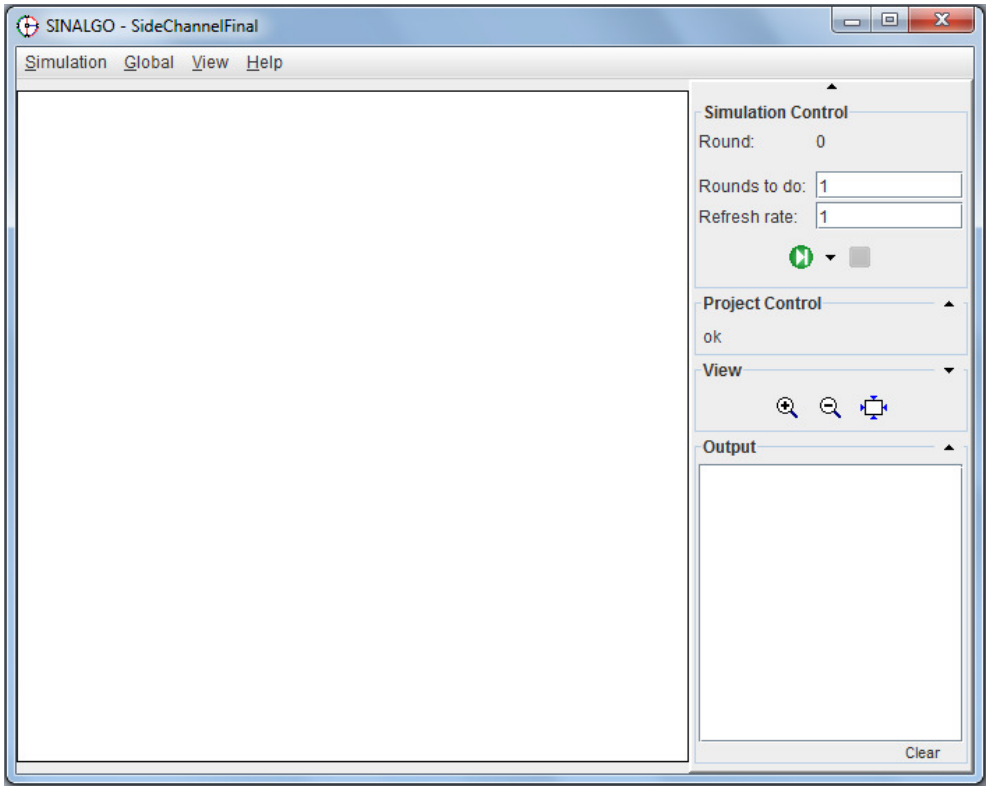

Figure 3: Sinalgo Simulation interface

Figure 4 shows the interface for controlling your simulation in Sinalgo. It consists of three sections.

- Top Menu
- Control Menu
- Canvas

The Top menu is designed to be accessed by the user to do some configuration. For example in the Simulation menu, one can generate nodes automatically or you can remove all the nodes from the system.

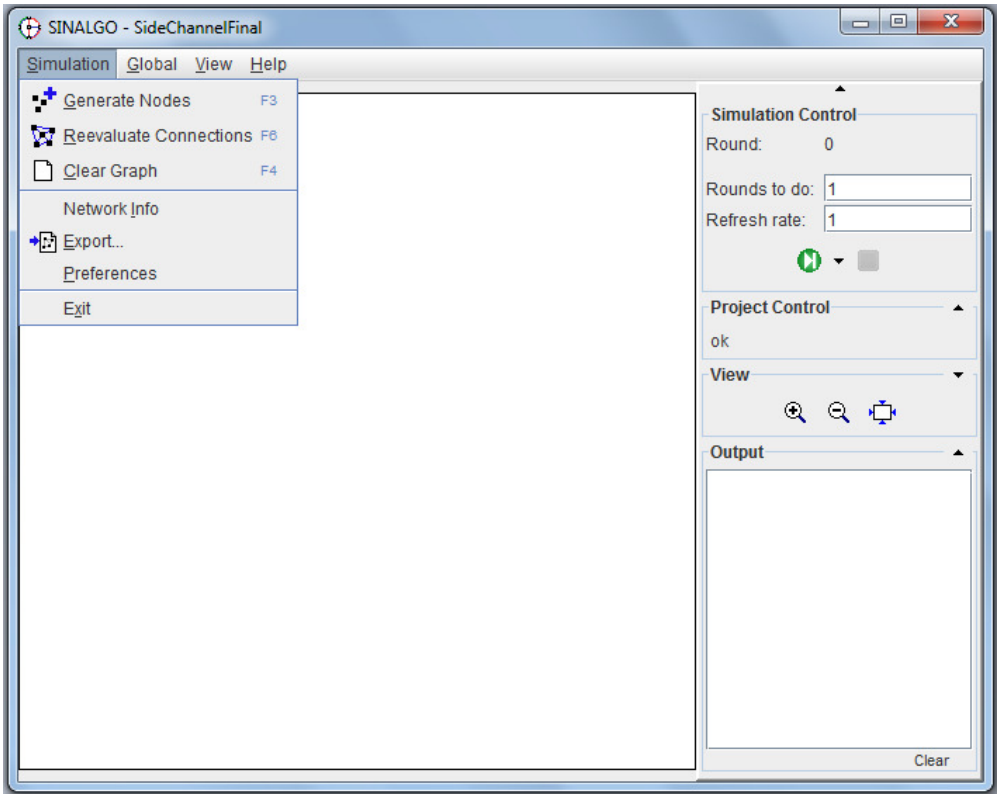

Figure 4: Sinalgo Simulation menu

You can also export the canvas view into PDF or PS format.

The Global menu is designed for the user to add new configurations. For example, in some scenarios, a feature was needed to save the current location of all nodes to a file so I can reload it multiple times without changing the position of the nodes.

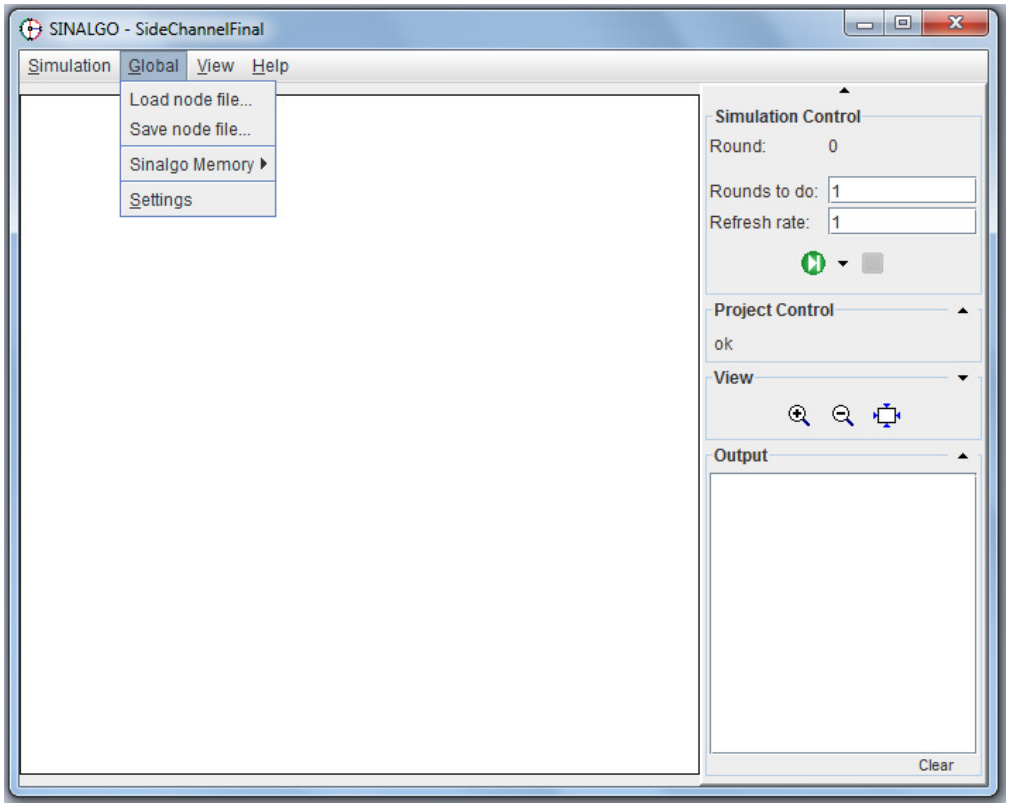

Figure 5: Sinalgo Global Menu

As illustrated in Figure 5, the two options "Load node file…" and "Save node file…" were created as helper tools that load and save the node configurations into files. The node configuration contains information such as type and location of nodes. Sinalgo offers a very good API for accessing this information. In order to add a new menu into that section, *CustomGlobal.java* file has to be modified and the following command has to be added before each method that needs to be called once the menu is selected. As an example, the following

code was added into CustomGlobal.java file which made Sinalgo create a " Save node file menu".

```
@AbstractCustomGlobal.GlobalMethod(menuText="Save node file...")
public void SaveFile() { }
```

CustomGlobal.java is a starter for each project. Each project has its own CustomGlobal.java file. This file contains all the Global features which the project needs for its own functionality. For example, if you want to share a static variable between multiple section projects, it has to be inserted into this file. This file is accessed by other program files of Sinalgo.

The other part of the simulator interface is the Control Menu. In the control Menu, start, stop and number of rounds can be controlled. Round is a smallest timestamp in the Sinalgo simulator. Each round consists of a set of operations. The time that it takes to finish one round could be different from another round. However the set of operations are similar. The set of operations are divided into two modes.

- Asynchronous

  In asynchronous mode, the simulation is driven by events. There are only two events which nodes react to: arriving messages and timer events. Thus, only the *Node.handleMessage* and *Timer.fire* are called. Also the global time of the simulation system is set to the time when those events happen before performing scheduled events. In asynchronous mode, mobility is not possible.

- Synchronous

  In synchronous mode, the simulation engine performs the set of actions and tasks for each round. Note that a single thread executes all tasks and actions so the execution is strictly sequential. Because of a single thread execution, no synchronization is needed to access global variables.

  The following list is the actions and tasks performed for each round which is explained in the Sinalgo Document [**1**]:

  - The framework increments the global time by 1.
  - CustomGlobal.preRound(); (Optional, project specific code. This method is called at the beginning of every round.)
  - The framework handles global timers that trigger in this round.
  - The framework moves the nodes according to their mobility models, if mobility is enabled.
  - The framework calls each node to update its set of outgoing connections according to its connectivity models.
  - The framework calls interference tests for all messages being sent, if interference is enabled.
  - The framework iterates over all nodes and calls *Node.step* on each node. The method 'step' performs the following actions for each node:
    - The node gathers all messages that arrive in this round.

        - *Node.preStep* (optional, project specific code. This method is called at the beginning of every step.)
        - If this node's set of outgoing connections has changed in this round, the node calls *Node.neighborhoodChange*
        - The node handles timer events that are called in this round. In other hand, every round, Sinalgo fires timer events which can be handles by every node.
        - *Node.handleNAckMessages* (handle dropped messages, if *generateNAckMessages* is enabled.)
        - *Node.handleMessages* (handle the arriving messages.)
        - *Node.postStep* (optional, project specific code, called at the end of each step.)
    - *CustomGlobal.postRound* (optional, project specific code, called at the end of every round.)
    - If *CustomGlobal.hasTerminated* returns true, the simulation exits.

The last part is the canvas which is a view port to your simulation. Every action can be displayed in canvas such as nodes, signal range, connections and mobility (moving node). Canvas also offers some custom menus for right mouse clicks. Figure 6 shows the custom menu which is made for custom simulation. This menu can be created by adding the following command before any methods.

@NodePopupMethod(menuText="Change to Malicious")

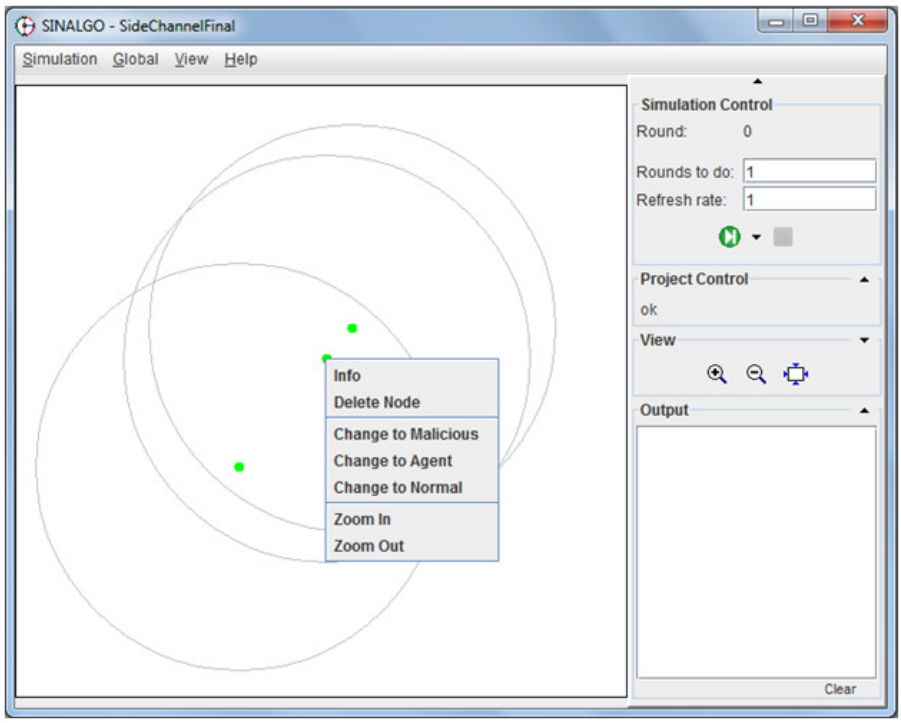

Figure 6: Sinalgo Canvas menu

The Canvas menu allows adding menus for each node. This feature helps to change any normal node into any type of nodes during the simulation.

3.2 Broadcast Communication Models

One of the main challenges with broadcast communications is deciding when to communicate in order to avoid node interference. Because Sinalgo does not support any particular MAC layer communication model (such as 802.11 CSMA/CA) one quickly observes that if all the nodes send messages during the same simulation round that a large majority of the received frames will be corrupted because of signal interference, resulting in all communications being ineffective. It is therefore necessary to come up with a simple communication strategy which

can minimize collisions among the nodes without having to re-create the 802.11 MAC layer in Sinalgo.

3.2.1 Uniform random distribution and Poisson distribution

A simple approach to reduce these collisions is to leverage a uniform random broadcast model. Some preliminary experiments showed at the end of this section, that the results of using a uniform random approach still led to a significant number of frame collisions.

Another distribution model that can be used to minimize potential collisions among wireless nodes is the Poisson distribution model because a large number of independent messages are being sent and each message has the same chance of occurring in this distribution. For example, the Poisson distribution was used by Kuntz et al. [**21**] to minimize collisions among beacons in an ad hoc wireless network.

The standard Poisson distribution model is shown in equation (3-1) where *k* is the number of occurrences (in our case this is the actual message transmission rate) and λ is the expected number of occurrences (in our case this is the desired message transmission rate *r*).

$$f(k,\lambda) = \frac{\left(\lambda^k e^{\lambda}\right)}{k!} \tag{3-1}$$

To generate a random Poisson value from uniform random values, the Knuth's algorithm [**22**] is used. To generate Poisson distributed messages among the nodes in the simulation, the value of λ is set to be fixed for all the nodes in the simulation (this number represents the rate of message transmission of each node and was set to 10, 40, and 100 in the simulation in different scenarios). At each simulation round, a random Poisson value *k* using Knuth's algorithm as well as a random uniform value *R* between *[0, 1]* are calculated. Then the value of *R* is compared

against the Poisson probability value of *k* which is calculated based on equation (3-1). If the random value *R* is less than the Poisson probability value of *k* then the node transmits a message, otherwise it does not. The following figure, Figure 7, represents the process of applying the Poisson distribution model to a communications model.

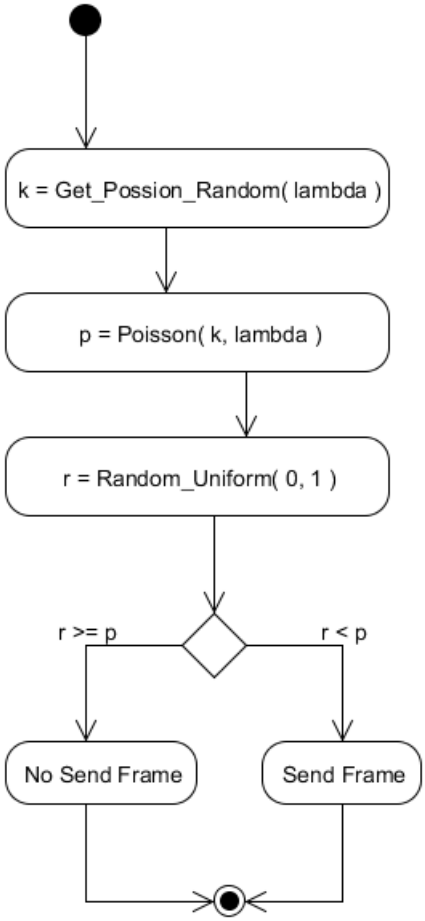

Figure 7: Apply Poisson for Communication

The specific scenario was designed to see the behaviour of the Poisson distribution model. The following is the setup of the simulation.

The above algorithm, which is shown in Figure 7, runs 100,000 iterations with different λ values such as 1, 5, 10, 40 and 100.

To validate the correctness of the Poisson generator used in the simulator, a number of graphs were plotted as shown in Figures 8 – 12. These figures illustrate the convergence of the Poisson generator as the average number of frames increases from λ = 1 to λ = 100.

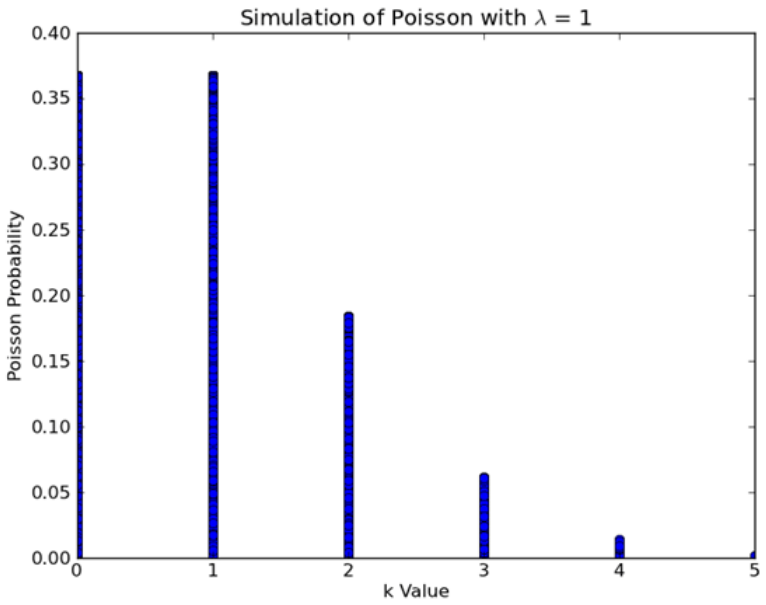

Figure 8: Simulation of Poisson behaviour with λ = 1

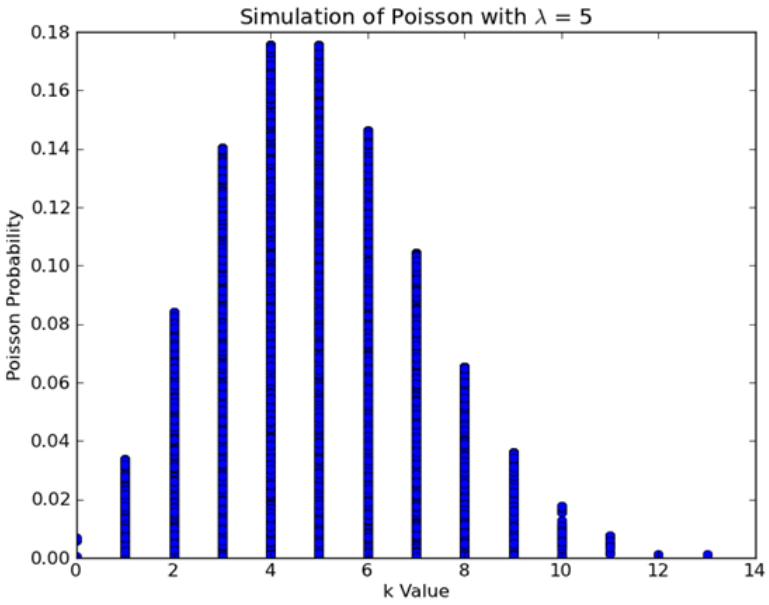

Figure 9: Simulation of Poisson behaviour with λ = 5

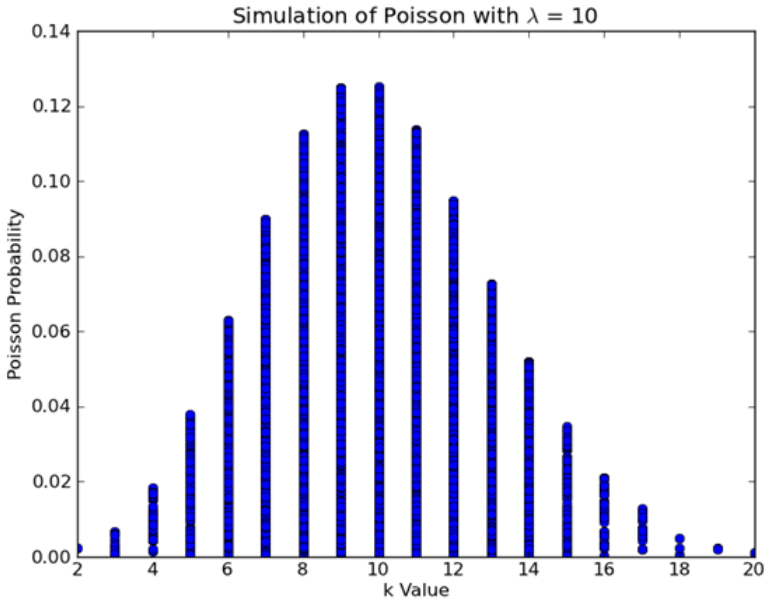

Figure 10: Simulation of Poisson behaviour with λ = 10

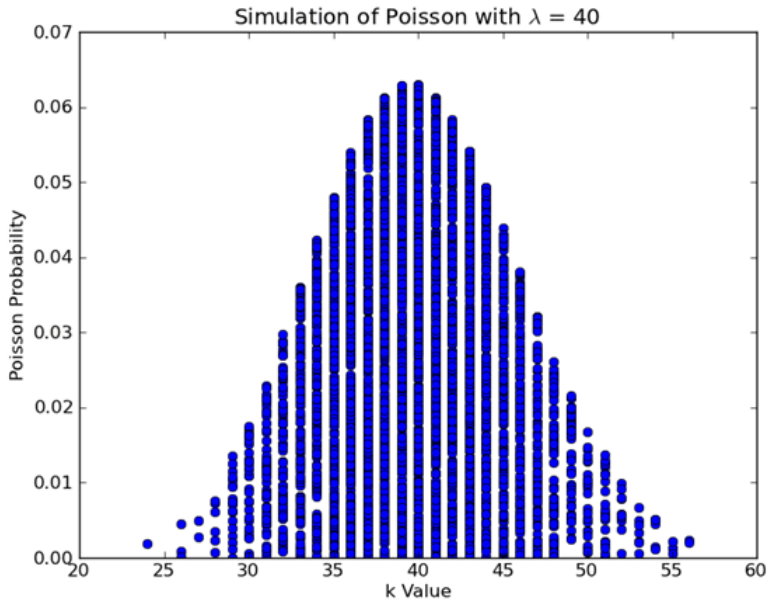

Figure 11: Simulation of Poisson behaviour with λ = 40

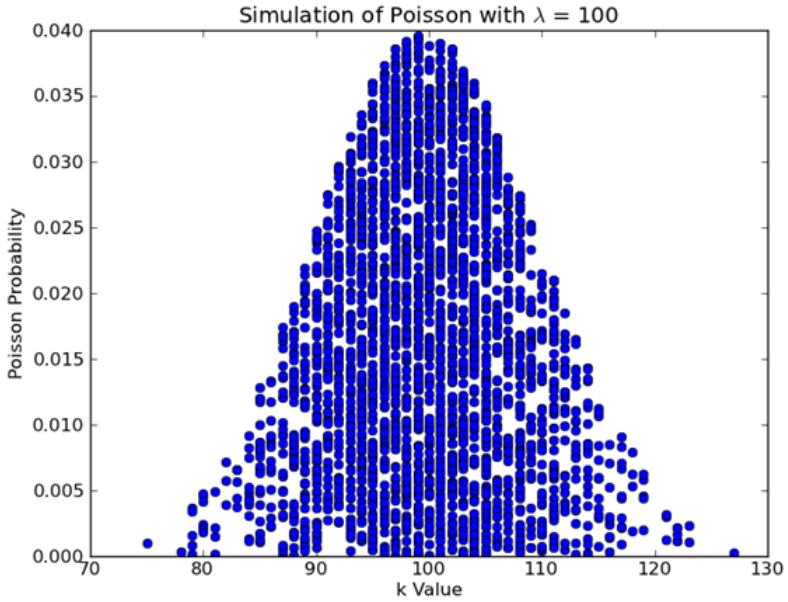

Figure 12: Simulation of Poisson behaviour with λ = 100

One thing worth mentioning is that the Poisson random generator always returns an integer which is why there are some gaps between each *k* value. In Figure 8 through 12, each blue dot indicates that a message is sent. As the λ increases, the bell curve is also shifted to the right.

Also by comparing Figure 11 and Figure 12, both have perfect bell curves, but the curve of Figure 12 with $\lambda = 100$ is meant to cover a larger *k* area than the curve of Figure 11 with $\lambda = 40$. Also as λ increases the Poisson function returns a smaller value. For example, Figure 12, with $\lambda = 100$, has a max *p* value of 0.04, which is smaller than that of Figure 11 with $\lambda = 40$.

In order to decide which of two (Uniform Random and Poisson) can be applied as a preliminary collision detection mechanism, both have to be tested in some scenarios.

The following Scenario is designed to test the behaviour of both Poisson and uniform random models.

The scenario consists of two types of nodes. The first type is called normal, which is transmitting data based on either the Poisson or Random model with a desired message transmission rate (*r*). The net effect is that the nodes are transmitting at a desired rate of communication *r* either in a uniform or Poisson distributed manner. The second type of node is called a sniffer. The job of a sniffer is to listen to network traffic and count the number of valid and invalid frames for each time. The simulation also runs multiple times by increasing the number of normal node. The experiment starts with two normal nodes at the beginning and increases with the following order, [2, 5, 10, 15]. The experiment runs three times with different transmission rates. Transmission rates are set to 10, 40 and 100 for every simulation.

Figure 13, shows the result of Poisson and Random with transmission rates of 10 frames per 1,000 rounds. The result indicates that as the number of normal nodes increases, the number of collisions increase. However, Poisson behaves really poorly with low transmission rates.

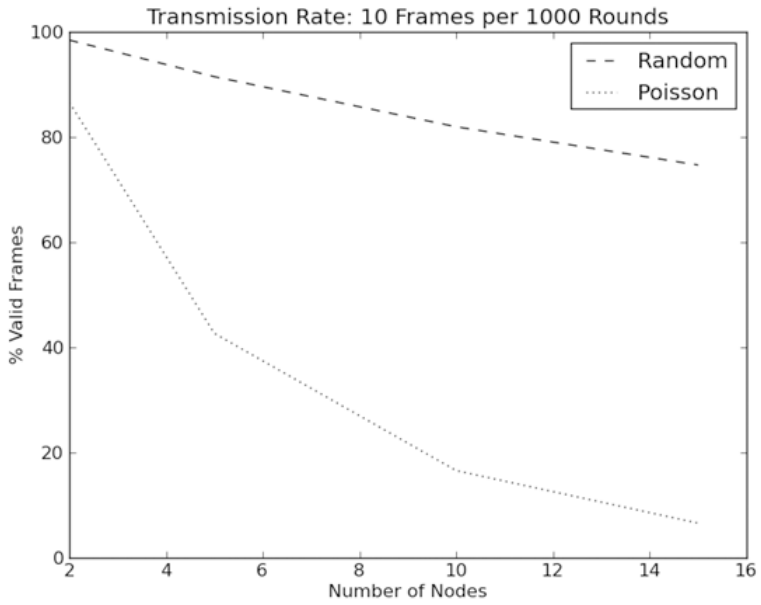

Figure 13: Poisson vs Random with 10 frames per 1000 rounds

Figure 14 shows the result of an identical scenario except for the transmission rate. In this simulation, the transmission rate was set to 40 frames per 1,000 rounds. The results show that by increasing the transmission rate, the Poisson model reduces number of collision frames than the random model compared to previous result.

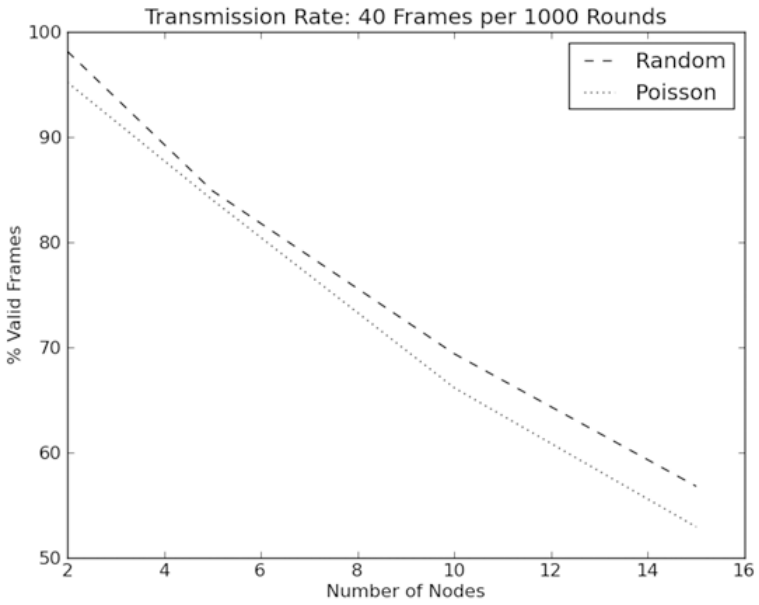

Figure 14: Poisson vs Random with 40 frames per 1000 rounds

The transmission rate was increased to 100 frames per 1000 rounds and the following figure, Figure 15, shows the result of that. As the transmission rate increases the Poisson starts to reduce the number of frames collision.

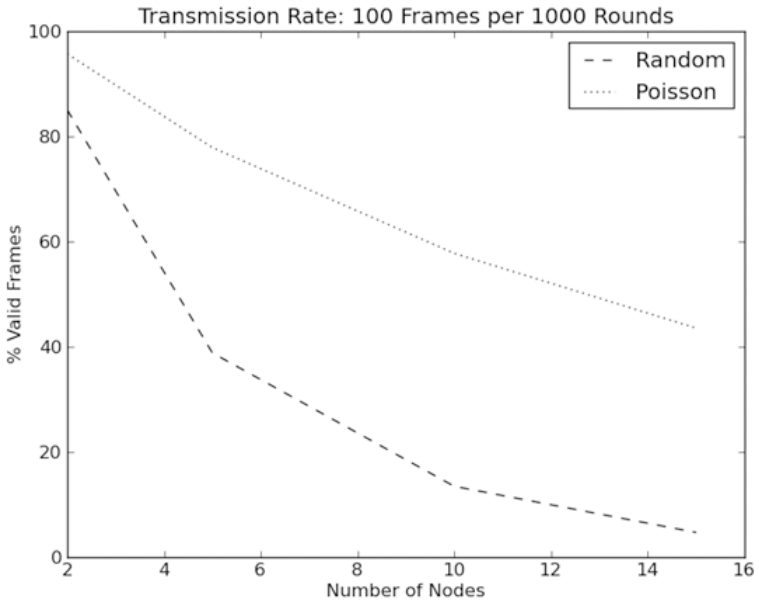

Figure 15: Poisson vs Random with 100 frames per 1000 rounds

As a result, for low transmission rate, random model reduces the number of frame collisions compares to Poisson model. However as the transmission rate increases and passes the 50 frames per 1,000 rounds, Poisson starts to reduce the number of frames collision.

As this section shows, in practice, the Poisson distribution reduces the number of frame collisions if the transmission is above 50 and this is the main reason why the Poisson distribution was chosen as the communication model among the nodes for this thesis.

3.3 Modifications to the Simulator to Support a Covert-channel

3.3.1 Passing frames with corrupt FCS

As it is described at the beginning of this chapter, almost all simulators considered in this dissertation, including Sinalgo, have a limitation which prevents nodes from receiving corrupted data. Sinalgo offers a better way to describe the wireless nodes which makes it possible to transfer corrupted frames to their destination. Sinalgo corrupts frames using Signal Interference plus Noise Ratio (SINR), which is a well-known interference model. SINR calculates a ratio $q = s / (n + i)$, which is the ratio of the received signal s and the sum of the ambient background noise n and the interference i caused by all simultaneous signals or transmission. The strength of the received signal is computed based on a simple exponential distance path loss model where the signal strength decreases exponentially based on the distance between the sender and receiving nodes. The transmission is successful if q is bigger than β, where β is a small threshold constant. However, if Sinalgo finds that any of those signals or frames is corrupted, it will remove the packet from the system.

The SINR class has a method called *isDisturbed(Packet p)*. This method has to be overridden from the base class. Every time this method is called, Sinalgo passes one packet into that method to see if that packet has to be dropped or not. So if method *isDisturbed(Packet p)* returns false, it means that the frame can be delivered to its destination, otherwise the frame is dropped. For our covert-channel purposes the algorithm was modified. This is done in such a way that *isDisturbed(Packet p)* always returns false. As Figure 16 shows, Sinalgo removes the packet from the queue list if the algorithm returns true.

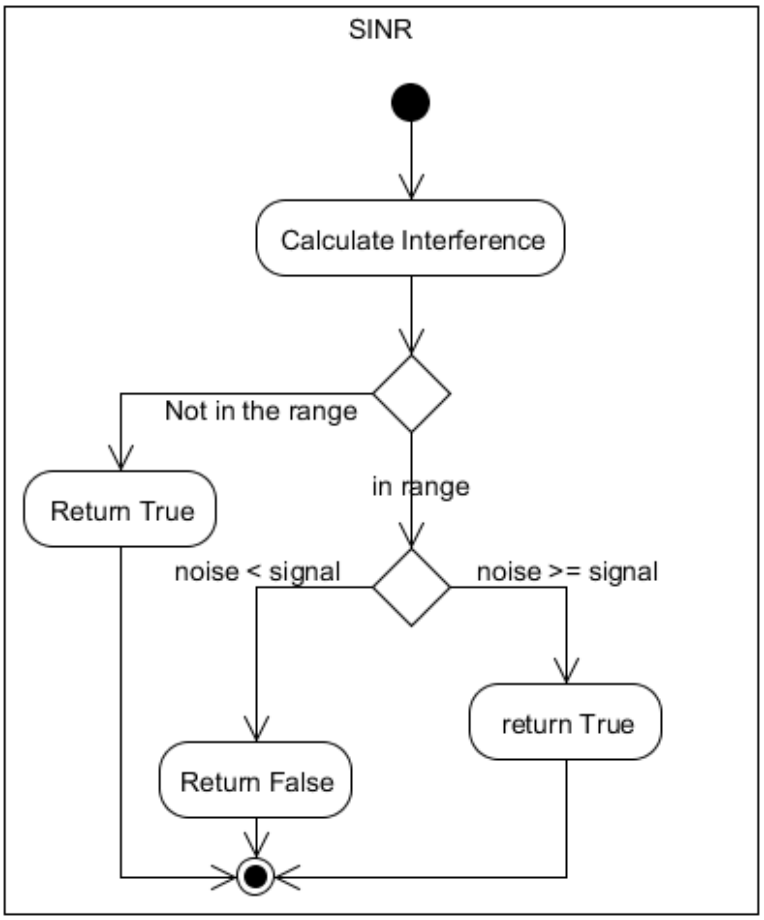

Figure 16: SINR original implementation

There are three sections which return either true or false. Each section is specific to its task. For example if the destination node is outside the range of the signal, the algorithm must return true to remove that packet from the system queue. If the destination node is in the range of that packet, the algorithm calculates the ambient noises and interferences of other packets based on the SINR equation, which we talk about in 3.3.1. If the summation of all noises and interference is less than the packet signal strength, then the algorithm returns false and the packet will be delivered to destination node safely; otherwise it drops the packet.

We redesign the original SINR implementation which is shown in Figure 17.

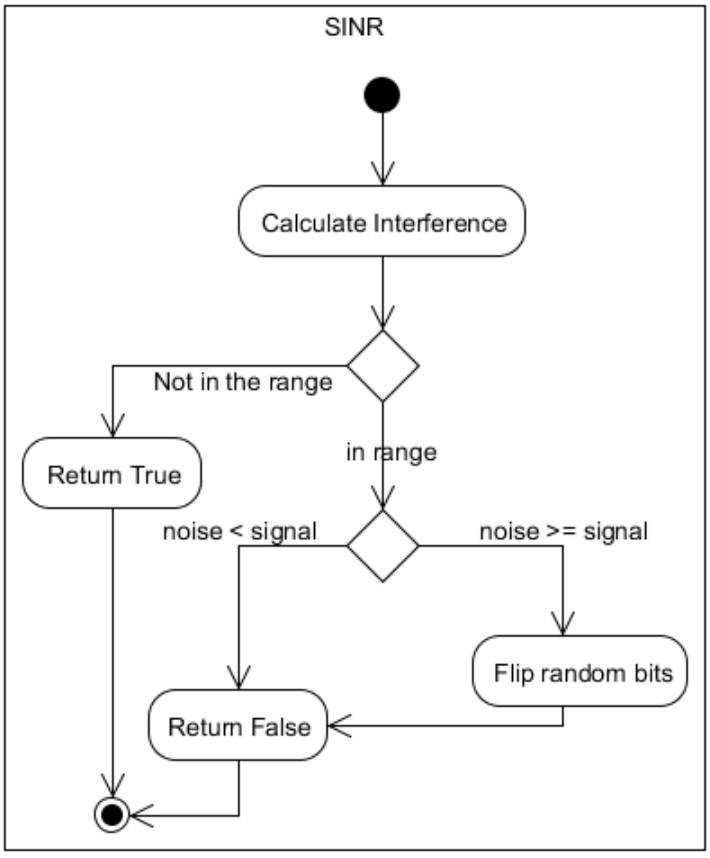

Figure 17: SINR re-designed implementation

The redesigned SINR algorithm is almost the same as the original version but with two changes. This new modification, as illustrated in Figure 17, still returns true if the destination node is not in the range of the sender node. However, it always returns false whether data is valid or invalid. In order to distinguish dropped frames from normal frames, a new module was implemented into the SINR algorithm. This new module flags those frames which are supposed to be dropped. Because in both conditions the algorithm returns false, the simulator has been forced to deliver all the frames. Now, since all frames will be transmitted and processed by the simulator, there was the need for a mechanism that allowed for determining whether a frame was valid or not. The approach was to flip a random bit in those frames which were supposed to be corrupted, and this was implemented in the block named “Flip random bits” of Figure 17.

The implemented module consists of some modifications to the message class and a new class called FCS. The FCS class is an implementation of the calculation of the FCS. It contains a specific method to convert any types of data into four unique hash bytes. FCS is a non-secure hash function; it is a non-secure hash, because it might produce the same four byte result from two different data. One of the main reasons for using FCS in frames and packets in the network is its fast calculation. Because of the huge amount of network traffic, the algorithm needs to be fast and reliable.

The other modification is on the message class. Every message in the Sinalgo simulator is inherited from *sinalgo.nodes.messages.Message* class. This class is a base class of all messages which pass by nodes in the Sinalgo simulator. Figure 18 shows the class diagram of FCS calculation and custom frame class which has payload and FCS hash value.

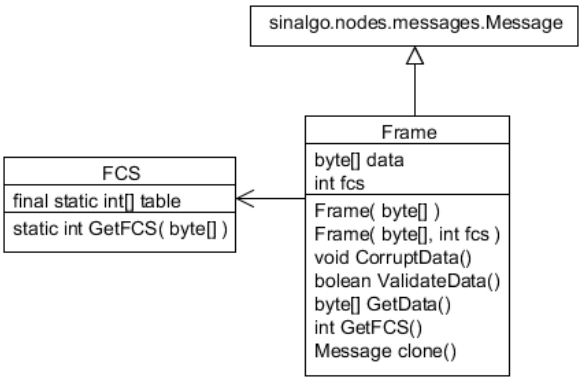

Figure 18: SINR modification class diagram

The following code shows the actual implementation of SINR in JAVA

```
//code-> calculates the signal
boolean disturbed = signal < beta * noise;
if(disturbed)
{
      if(p.message instanceof Frame) ((Frame)p.message).CorruptData();
}
return false;
```

As the above code shows, if the signal value is less than overall noise, the code will execute the second condition and if the message of the packet is an instance of the Frame class, the message object type is cast into Frame object and then calls the *CorruptData* method. *CorruptData* basically corrupts the data in a random location. The reason for that is adding randomness into corruption of data makes it more realistic, since in real scenarios bits are flipped in random locations.

3.3.2 Introducing Agent Node types

In order to have a proper and realistic simulation, one more thing had to be introduced into the Sinalgo simulator. Normal nodes are one part of the simulation. Therefore, by just having normal nodes, the covert-channel communication cannot have a proper and realistic situation. In order to have one, malicious nodes have to be introduced into the simulation. Malicious nodes in Sinalgo are just like regular nodes, except that they create a covert-channel communication between themselves.

There is only one thing left. In the real world, monitoring the behaviour of all nodes is unrealistic. What can be done is monitor of the traffic of the network. To accomplish this, new kinds of nodes, called Agents, need to be introduced into the system. These nodes listen to traffic and gather information and intelligence from neighbour nodes. These nodes are not simple sniffers; they are actually something more. They can share information between themselves and by having the feature of sharing information; they can apply collaborative anomaly detection techniques such as Pearson and Spearman correlation coefficient (described in more detail later in Chapter 5).

3.3.3 Processing the Captured Data

Data processing is the main mechanism for finding traffic anomalies. Each agent node gathers a massive amount of data which is saved into a file for further processing. In the end, each agent will generate its own file which contains all the gathered information.

Processing all information coming from agents would be an important part of this thesis. This is why the Python programming language [**23**] was chosen for that task. Python is flexible, fast

and simple to use. Also it is supported by huge developer communities and offers various libraries where all the graphs are generated from one of its libraries called matplotlib [**24**].

Each agent node stores all collected information into a unique file where it is named after its own ID. The structure of each agent file is described as follows:

Each agent records information of each round and stores it into a row. Each row consists of three columns. The first column is the total number of valid frames received. The second one is the total invalid received frames and the last column is the total of all received frames.

Each simulation ran for 100,000 rounds, so at the end of each simulation, each agent has 100,000 records of data. Creating and making a meaningful result from each data file requires a set of actions and processes. Python reads all the data in each agent file and divides 100,000 rows evenly into 100 rows of 1,000 records. Each 1,000 record is added together and placed into an array. In the end, the data is converted from 100,000 records into 100 elements, with each element representing the summation of 1,000 records. This summation creates sample data which makes more sense compared to the original data. For example, the following set of data is captured from one agent during one simulation.

*Agent*_1 = { [0,0,0], [0,0,0], [0,0,0], [0,0,0], [0,0,0], [1,0,1], [0,0,0], [0,0,0], [1,0,1], … }

As can be seen from the above data set, each block of data represents a round and contains three columns. The first column represents the number of valid frames that is captured during a specific round. The second column shows the invalid frames and the last one shows the total.

Because the data is so wide, finding anomalies would be extremely difficult. The following figure, **Error! Reference source not found.**, shows the structure of the agent file.

Figure 19: File Structure and Process data schema

Chapter 4: Anomaly Detection

4.1 Threat Model

In our model an adversary is capable of a) physically position itself in the network, b) inject purposely corrupted frames and understand malicious frames for establishing covert-channel communication. In this dissertation we call adversary nodes “malicious” nodes. Furthermore nodes in our network which are not malicious are capable of transmit normal messages back and forth.

Thus, from a defence point of view, it is possible to “drop” agent nodes in the network which exhibit the following capabilities: a) can gather statistical information about network traffic, and b) can communicate with each other to share statistical information.

Finally, and again from a defence perspective, what we want to achieve is detection of the presence of covert-channel communication in the network.

4.2 Overview

Statistical techniques can be used to identify anomalies in any set of data. The problem approached in this dissertation follows a similar concept. The simulation generates a huge data set which needs to be analyzed in order to identify anomalies. The idea is that these anomalies are the result of manually injected corrupted frames. Those corrupted frames must be identified and classified as a threat to the system. As explained in the previous chapter, this kind of attack could be used to transfer data covertly from one location to another. In order to identify the problem, some algorithms have to be tested to see if they can detect those anomalies.

The first thing that comes to mind is to use simple approaches such as mean and percentage of overall received versus sent frames. For the purposes of this work, our data is composed of two kinds of frames:

a) Valid frames. Valid frames are those frames which are received without any errors, i.e., with a valid FCS.
b) Invalid frames. Invalid frames in other hand, are those frames which are received with errors, i.e. with an invalid FCS. Invalid frames are categorized into three different types:
    i. Normal invalid frames
    ii. Malicious frames
    iii. Invalid malicious frames

Normal invalid frames are those normal frames that, because of various reasons such as interference and the shadowing effect, become corrupted and thus invalid. For example, in a noisy environment, valid frames become invalid more often than usual. These corrupted frames are also called naturally corrupted frames. The following figure, Figure 20, shows the normal transmission, normal error transmission, and covert-channel percentage based on simulation data.

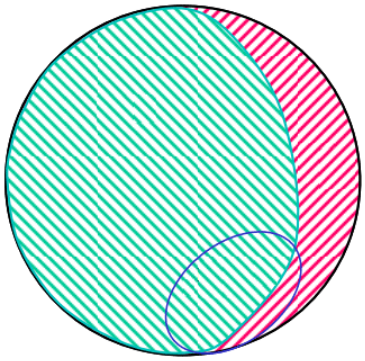

Figure 20: A Venn diagram from Marvin et al. [7] which shows the type of frames in a wireless channel. Green indicates normal behaviour. Red indicates normal error and the blue partitioned area is the covert-channel.

Malicious frames are those frames which are injected by malicious nodes; malicious nodes are those nodes which are participating in a hidden channel communication by tweaking their FCS algorithm to produce the desired FCS value, into the system, i.e., nodes that intentionally corrupt the FCS with the intention of establishing a hidden channel. Those frames are not invalid and as a matter of fact those frames contain valid information but they have been marked to be seen as invalid. A simple anomaly detector sees those frames as invalid and simply ignores them.

Invalid malicious frames are those which are corrupted during the process of transmission, i.e., a malicious frame can be corrupted because of a noisy environment. The probability of a malicious frame being corrupted due to the noise, such that it becomes viewed as a valid frame is 2 to the power of 32. The reason for that is that the number of available bits for FCS is 32 bits.

4.3 Percentage and Mean Value

The percentage and mean value could be used as first lines of defence and detection to attempt to identify these anomalies. This can be described by the equation (4-1).

$$f(V_i, T_i, n) = \frac{1}{n} \sum_{i=0}^{x} \frac{V_i}{T_i} \quad (4\text{-}1)$$

where the formula calculates the mean value of utilization of valid frames over the total number of received frames, and V_i represents valid frames, T_i represents total received frames and n represents the number of frames. The equation gives us an overview of the system behaviour. If a history of normal usage of the wireless network is provided, it could determine what normal behaviour is for the system and then compare this normal behaviour with the current situation. The likelihood of false positives and false negatives while using this approach is high since the mean value cannot shows the range of numbers in the set of data. For example, the mean value of the following two sets of numbers are the same but the range of numbers are not.

$X_1 = \{1, 99, 100\}$ $\bar{x} = 66.6$

$X_2 = \{66.6, 66.6, 66.6\}$ $\bar{x} = 66.6$

4.4 Variance and Standard Deviation

In the previous section, the percentage and the mean value were proposed to find anomalies. Unfortunately this approach lacks accuracy for detecting anomalies. In terms of detecting anomalies, using more sensitive approaches is needed. Variance and standard deviation could be used; these two approaches are highly sensitive to relatively drastic changes. As an example, if some fluctuations happened in the number of invalid frames over certain period of time due to the presence of purposely corrupted frames, the variance and standard deviation will show

those with significant spikes. For example, the following two data sets have the mean value of 100.

$X_1 = \{100, 102, 105, 98, 95, 100\}$, $\bar{x}$ = 100, Variance = 11.6

$X_2 = \{50, 105, 95, 98, 102, 150\}$, $\bar{x}$ = 100, Variance = 1011.6

However, X_2 has 2 values which have the big jumps. These jumps can be captured by calculating variance.

4.5 Pearson and Spearman

One of the most familiar measures of dependence between two sets of data is the Pearson product moment correlation coefficient described by equation (4-2). It is also called Pearson's *r* correlation. This measurement is obtained by dividing the covariance of the two sets by the product of their standard deviation. The Pearson correlation can only be defined if both of the standard deviations are finite and both of them are not zero.

The resulting value of Pearson's *r* correlation is in the interval [-1, 1]. Pearson's correlation coefficient can also become zero if the variables are independent from each other. The converse of the previous statement (i.e., if the correlation coefficient is equal to zero then the variables are independent from each other) is not a true statement since the linear dependencies between two variables can only be detected by the correlation coefficient. For example, if *X* is a set of random variables which are distributed symmetrically and the *Y* set is calculated from the formula: $Y_i = X_i^2$, then *X* and *Y* are dependent since *X* determines *Y*. Since these two sets are not correlated, their correlation becomes zero.

If a set of *n* data elements of *X* and *Y* is given, which are written as x_i and y_i where *i* increments from zero to *n* - 1, then the sample can be used to calculate the Pearson correlation *r* which is between *X* and *Y*. Equation (4-2) can be used to find the correlation coefficient of sets *X* and *Y*.

$$r_{xy} = \frac{\sum x_i y_i - n\bar{x}\bar{y}}{(n-1)s_x s_y} \tag{4-2}$$

where $\bar{x}$ and $\bar{y}$ are the sample means of *X* and *Y*, and s_x and s_y are the sample standard deviations of *X* and *Y*, then the correlation of two sets of data will be 1 if they have a perfect line with a positive slope as shown in Figure 21. On the other hand the correlation of two sets of data will be -1 if a perfect line can been drawn and the slope of that line is negative as illustrated in Figure 22. The last example is shown in Figure 23, which has a result of zero correlation because it is very hard to draw a line between those dots.

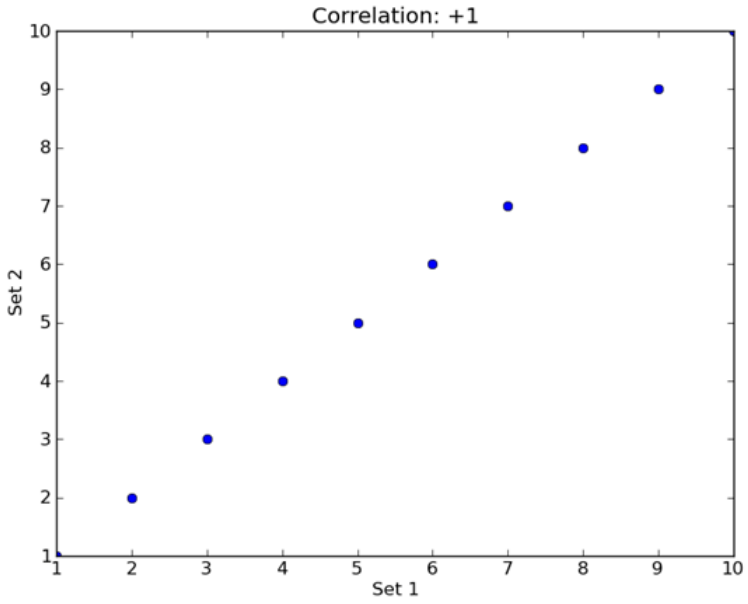

Figure 21: Correlation +1 for two sets

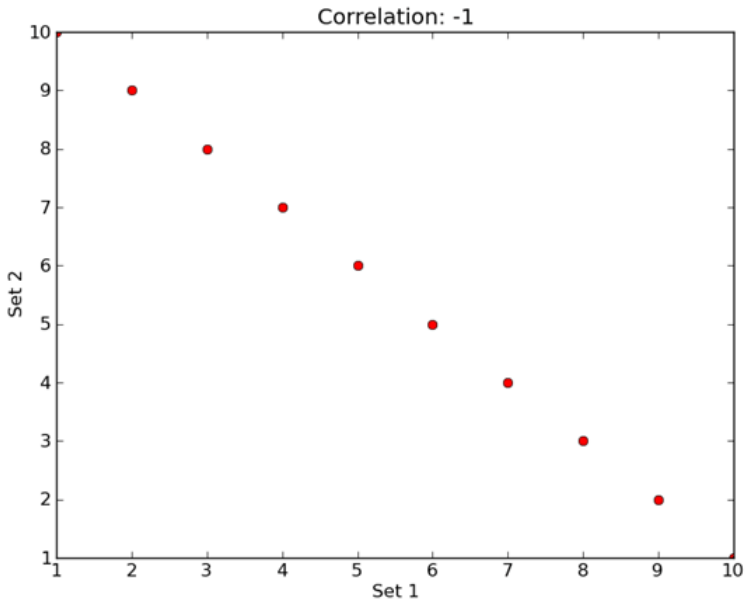

Figure 22: Correlation -1 for two sets

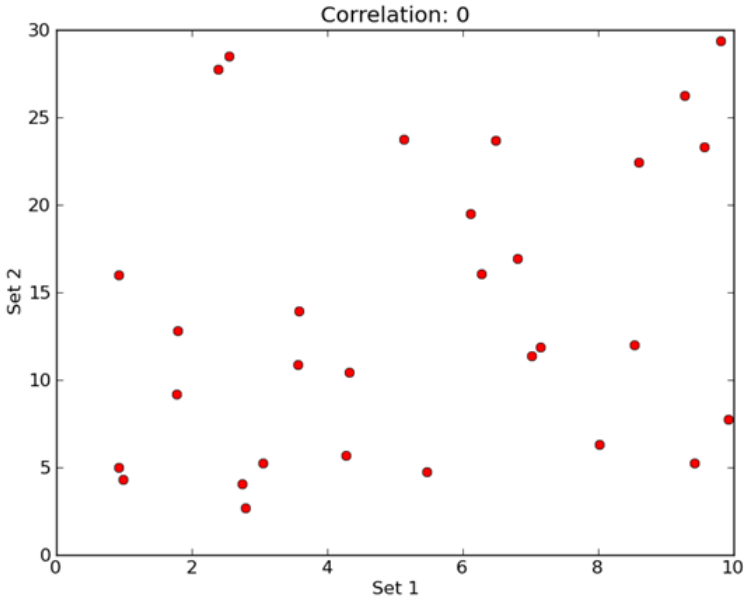

Figure 23: Correlation 0 for two sets

Chapter 5: Experiment Results and Evaluations

This chapter explains the two anomaly detection mechanisms used in this dissertation, which are based on statistical approaches for identifying anomalies in wireless communications. These two approaches are applied in simulated scenarios using Sinalgo. In this research context define an anomaly is defined as network activity with an abnormal proportion of invalid frames, which may indicate the presence of covert-channel communication.

The focus of this chapter is to investigate the accuracy of using single agent and multi-agents systems. Therefore, this chapter is divided into two separate sections. The first section applies a single agent system into multiple scenarios and compares them with each other. The second section focuses on a multi-agent system.

5.1 Single Agent System to Detect Covert-channel Communications

Early in our experiments we realized that the variance promised to provide a more sensitive measure as opposed to the mean value for detecting anomalies given our particular scenarios. While this statement needs further validation, we considered that variance would provide a good metric for our particular objectives.

The single Agent System is designed to work based on historical data. History is a set of data collected during a period where the system exhibits normal behaviour, i.e., without the presence of covert-channel communication. Historic data are also separated into two sections. The first section is the valid frames statistical information and the second one is the invalid statistical information. Having these two sections will help the Single Agent System to detect anomalies. Figure 24 shows the new technique which is implemented to detect anomalies

based on variance tables which we called Variance-based Single Agent Anomaly Detection algorithm (VSAAD).

As the figure shows, the first step is to read normal behaviour of network which is collected by an agent for some time period. Data is then processed by the next element which calculates and generates variance tables for both valid and invalid data gathered by the agent node. Then the algorithm finds the upper and lower bound for each table. Reading new data will begin right after constructing upper and lower boundaries. So from now on if the new variance which is calculated from the new data set is between both boundaries, it is deemed that there is no anomaly. If one of them detects anomaly, the algorithm triggers an alarm.

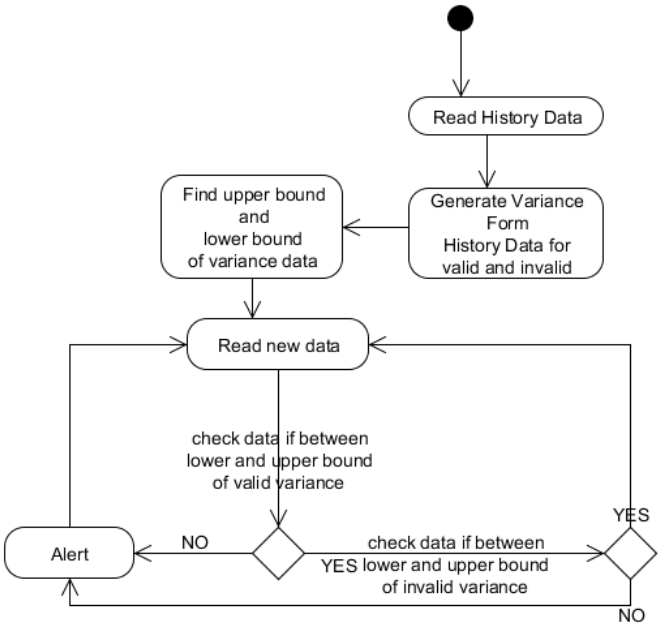

Figure 24: Variance-based Single Agent Anomaly Detection algorithm (VSAAD)

To test the effectiveness of the Single Agent System algorithm, a number of scenarios are created. These scenarios have couple of variables which are needed to be configured as follows:

- Transmission rate (λ) of normal nodes and malicious node
- The number of active normal nodes
- Maximum number of malicious frames
- Maximum number of rounds for each scenario

Because of effectiveness of transmission rate (λ) of malicious frames in each scenario, λ becomes variable and set to 10, 40 and 100. Also in Single Agent System, all the nodes are in range of each other, so position of the nodes does not important. However, the number of active normal nodes is an important factor. Thus, the number of normal active nodes is set to 2, 5, 10, and 15 for each scenario. Each scenario runs for 100,000 rounds. The number of rounds is set to 100,000 to allow for sufficient rounds to reach a steady state both after starting the simulation and after finishing the establishment of the covert channel [35, ch.25, p.423]. Table 1, shows that from round 1 to rounds 49,999 the agent gathers data as a history data, because there are no malicious transmission happening. Based on Table 1, the scenarios are configure to increase number of malicious frames every 10,000 frames. For example, between rounds 50,000 and 59,999, only 10 malicious frames will be injected into the system by the malicious node. The numbers of malicious frames shown in Table 1 were chosen arbitrarily so that they reflected gradual increases of malicious frames from what we thought would be negligible to a larger number of frames that we considered would be easily detectable.

Round	# Malicious frame
50,000 - 59,999	10
60,000 - 69,999	40
70,000 - 79,999	100
80,000 - 89,999	200
90,000 - 100,000	1000

Table 1: Assign malicious frame rate to each round

So as a result, the following setup is created to investigate the behaviour of Single Agent System.

- Normal Transmission rate set to 40 and malicious transmission rate set to λ = 10, 40,and 100
    - Number of active nodes is set to n=2, 5, 10, and 15.
        - Maximum simulation rounds set to 100,000 rounds
            - Runs the same simulation 32 times

Jain [24, ch.13, p.206] explains that a "reliable" confidence interval can be obtained with large samples and suggest to use samples of size greater than 30. Therefore, we used 32 samples in the simulations.

The above configuration creates three categories. Each category is divided into three sections. Each section contains 32 experiments. As discussed at the beginning of this section, the VSAAD in Figure 24 tries to create two separate tables, namely the Valid Variance Table and the Invalid Variance Table, for finding upper and lower boundaries for normal behaviour. Once these boundaries are found, the algorithm tries to detect anomalies.

The total number of simulations for the above configuration is 384. This number comes from the following formula.

3 sections x 4 categories x 32 experiments = 384 simulations.

Category refers as transmission rate, which is changed for every scenario. The value for transmission rate is 10, 40 and 100. Section refers as number of active normal nodes in scenarios. Each category has 4 sections. Each section represents the number of active normal nodes. In that case, the number of active normal nodes is set to 2, 5, 10 and 15.

For the purpose of demonstrating on how Single Agent System works, Figure 25 shows a snapshot of one of the 32 simulations which has two normal nodes, one malicious node and one agent node. Each simulation ran for 100,000 rounds.

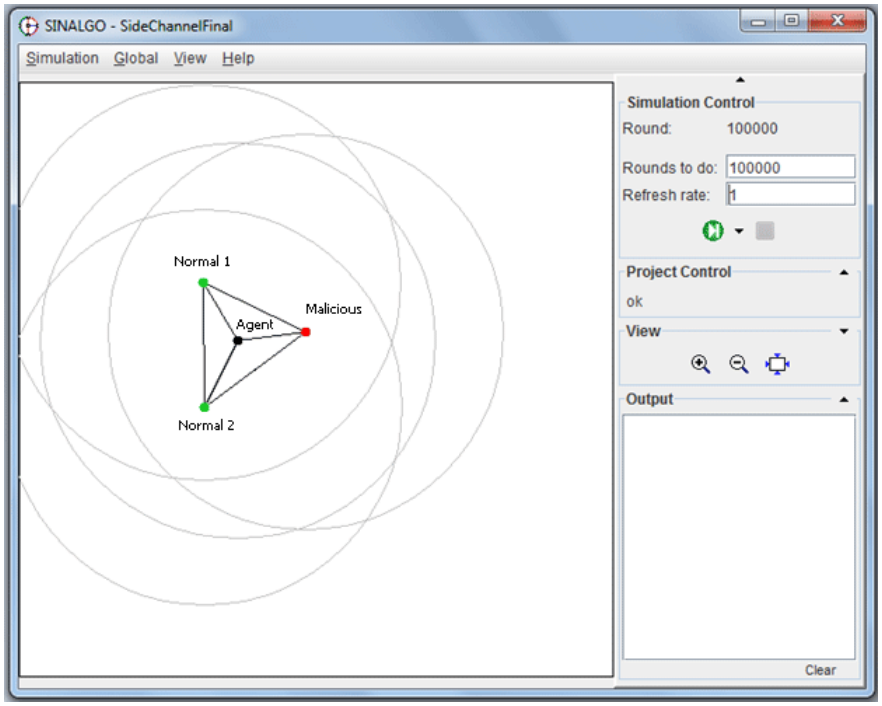

Figure 25: Snapshot of running two normal nodes, one agent and one malicious node

Table 2 and 3 show the processed data with means and variances for both valid and invalid frames for a single experiment. The yellow-highlighted rows indicate when the malicious node injected purposely invalid frames.

Rounds	Mean	Variance
0-10 k	125.400000	120.040000
10-20 k	126.400000	106.440000
20-30 k	124.500000	123.450000
30-40 k	120.300000	96.010000
40-50 k	121.400000	113.040000
50-60 k	119.000000	149.400000
60-70 k	121.400000	61.840000
70-80 k	121.400000	92.040000
80-90 k	118.900000	90.490000
90-100 k	117.000000	55.800000

Table 2: Means and variances for valid data in experiment one with two normal nodes and one malicious node

Rounds	Mean	Variance
0-10 k	12.700000	25.210000
10-20 k	10.600000	12.040000
20-30 k	11.900000	16.090000
30-40 k	10.600000	9.640000
40-50 k	14.400000	37.440000
50-60 k	14.700000	36.610000
60-70 k	17.500000	348.650000
70-80 k	21.000000	215.600000
80-90 k	32.400000	454.040000
90-100 k	58.700000	70.210000

Table 3: Means and variances for invalid data in experiment one with two normal nodes and one malicious node

Figures 23 and 24 plot the result of applying the algorithm in Figure 21 based on above tables.

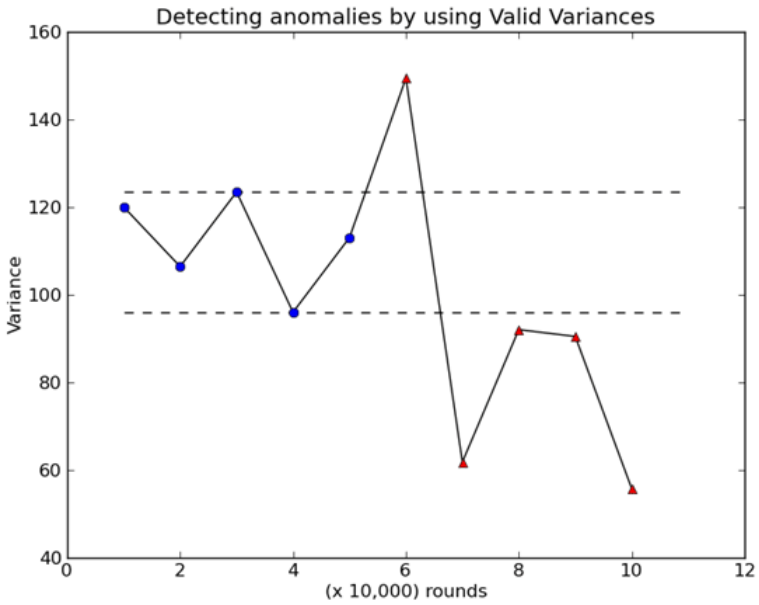

Figure 26: Single Agent Algorithm apply to experiment one with valid variance table

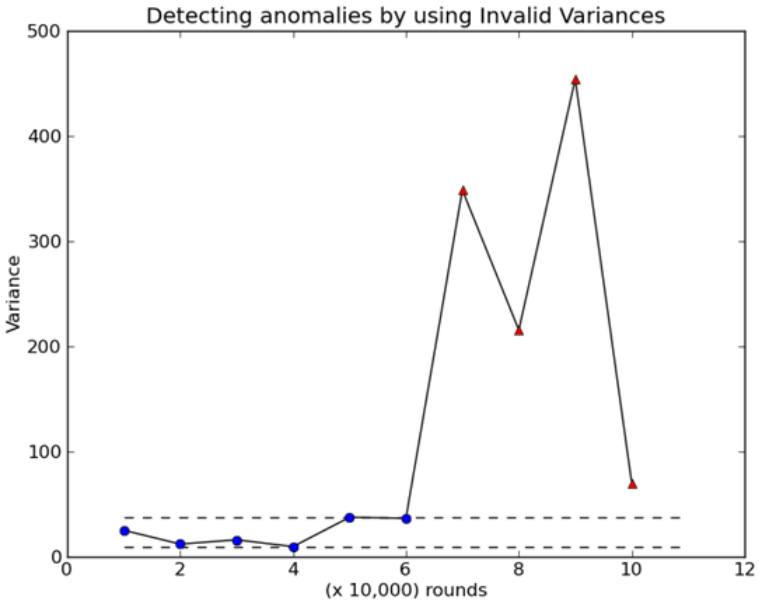

Figure 27: Single Agent Algorithm apply to experiment one with invalid variance table

As the figures show, the red triangles indicate the anomalies which are detected by the algorithm. As depicted by Figure 27, at round 60,000, no anomaly is detected since the value is

within the boundaries. However in Figure 26, the anomaly is detected and represented with a red triangle to indicate that it is out of the threshold boundaries.

As was discussed at the beginning of this section, all of the experiments differ in that the number of normal nodes increases gradually from one experiment to the other. All of these experiments are designed to investigate the behaviour of the Single Agent Algorithm listed in Figure 24.

Figure 28, Figure 29, and Figure 30 show the result of all these simulations. Each figure shows the detection rate for each transmission rate (λ). The figures also contain the detection rate of using one table, either Valid Variance Table or Invalid Variance Table, and both tables together. While further experimentation will be needed to corroborate our observations, the results seem to suggest that by using both tables the detection rate becomes higher which makes it suitable for applying for Single Agent System. The number above each red triangle in Figure 28 28, and 29 indicates the 90% confidence interval for the mean value of detection rates for 32 simulations. For example in position [2, 90], the 5.20 indicates the error rate of detection.

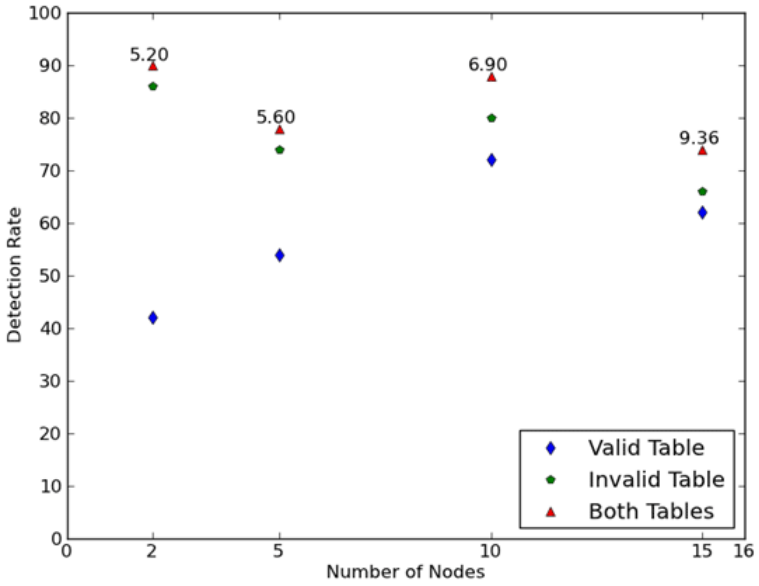

Figure 28: Detection Rate vs. Number of Nodes for λ = 10

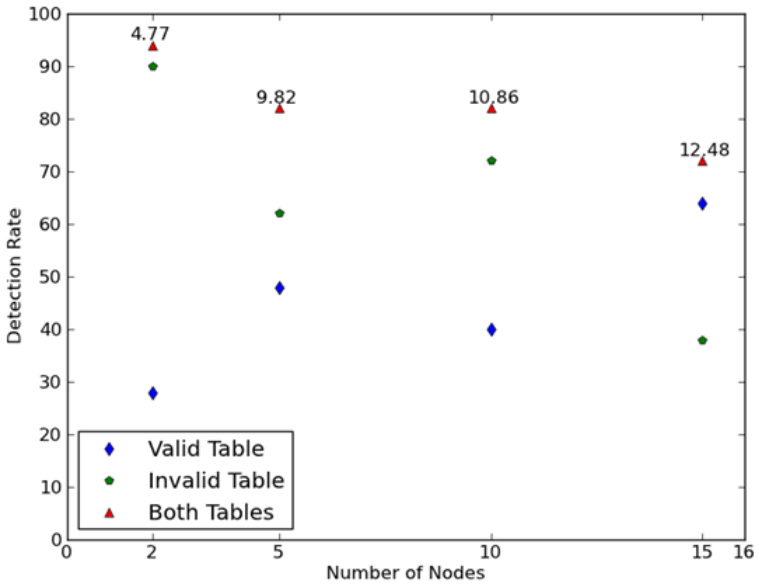

Figure 29: Detection Rate vs. Number of Nodes for λ = 40

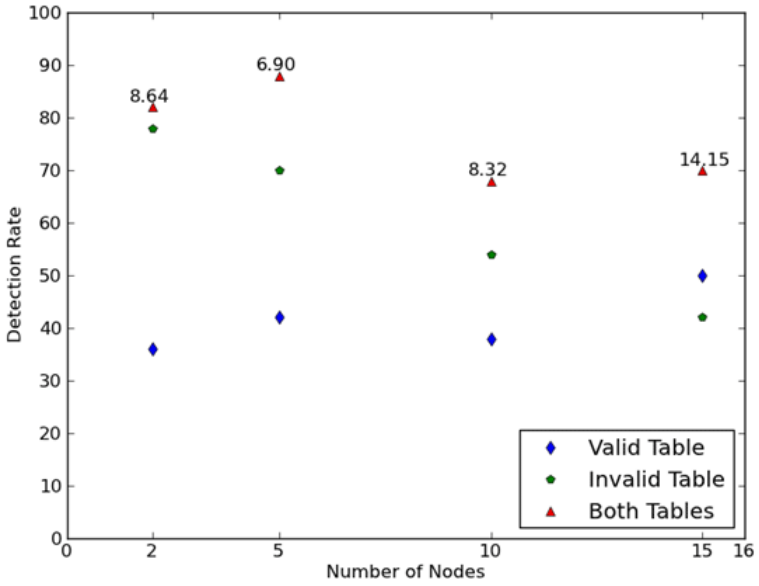

Figure 30: Detection Rate vs. Number of Nodes for λ = 100

Even though the number of active nodes is increased, the detection rate states above 70% which makes it suitable for detecting this types of anomalies.

5.2 Multi Agent System to Detect Covert-channel Communications

In this section, a Multi Agent System is introduced for the detection of covert-channel communications. The Multi Agent System has some advantages over the Single Agent System. For example, this method uses multiple agents to listen to traffic and gather statistical information, which makes it more accurate than the Single Agent System. They also can cover a larger area of the network. Figure 31 shows the multi agent system coverage area.

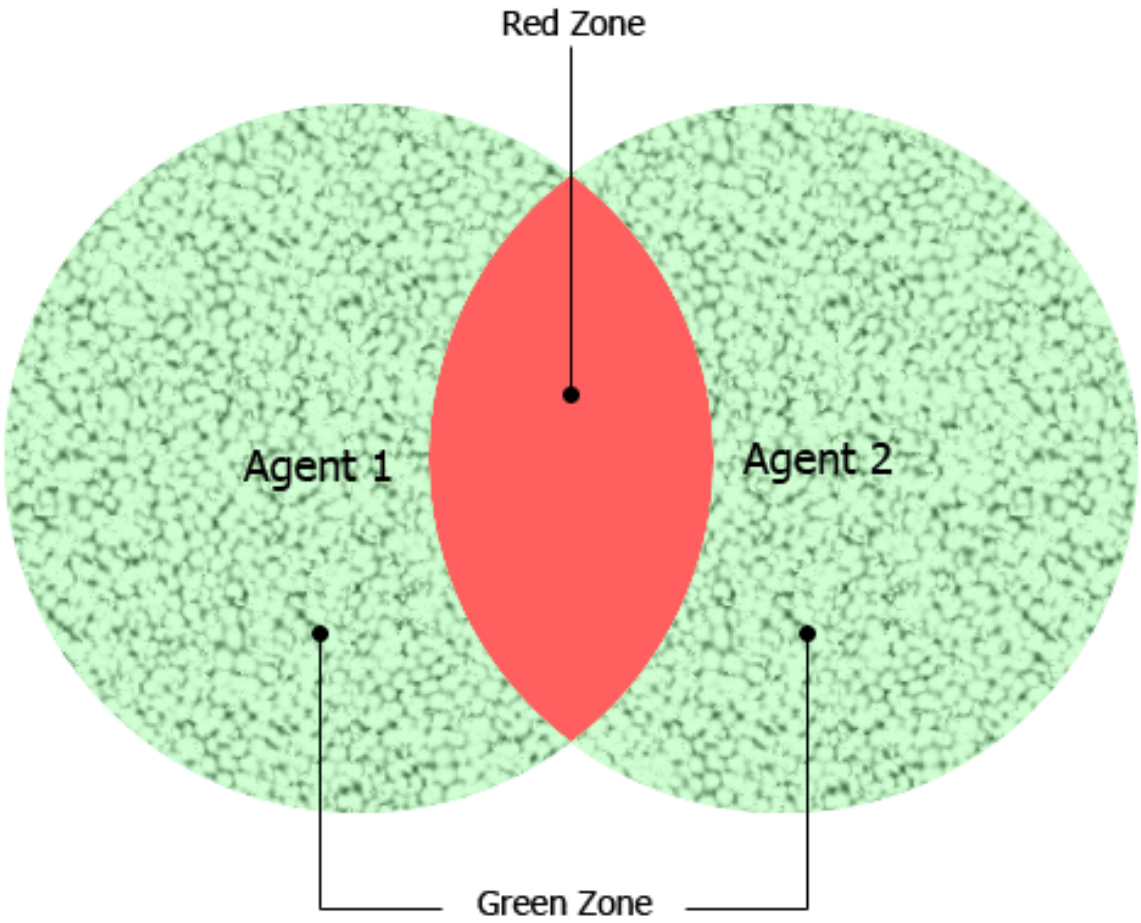

Figure 31: multi agent coverage

In a multi agent system, multiple agents are placed in different locations to cover a wider area. This technique brings a new advantage over the single agent. As Figure 29 shows, the area is divided into three sections. The green-coloured area represents an area that is reachable by only one agent while the red-coloured area represents an area reachable by two agents. The remaining white-area is not reachable by any agent.

In order to investigate the behaviour of multi agent system, a specific scenario must be implemented and designed. As described in Section 5.1, there are a number of parameters which affect the result of the simulations. Those parameters are listed as follows:

1. Location of malicious node.
2. Number of active normal nodes in the green-coloured and red-coloured areas distributed randomly.
3. Number of agent nodes.
4. Transmission rate of normal and malicious nodes.

The scenario uses two agents in the fixed location. Therefore, the scenario is divided into two major sections determined by the area where the malicious node is located with respect to the zones depicted in Figure 30. The number of normal active nodes is changed from 5 to 15. The normal nodes are placed in the area, green-coloured and red-coloured area, with the random distribution model in each simulation. The number of simulations is set to 32. Also the transmission rate for both normal and malicious nodes is set to $\lambda = 40$.

Figure 32 is plotted from both agents' data as an example. Those data which are gathered from both agents are spliced into 10 segments of 10,000 round. Each segment is divided into 10 groups of summation of 1000 rounds. This process is shown in Figure 18.The following figure, Figure 32, is plotted from the mean value of the invalid frames for every segment. Based on preliminary experiments, it was decided to choose mean as primary equation because it shows a pattern for normal behaviour. The five blue dots indicate the normal behaviour as from round 1 up to 50,000 there is no malicious activity happening. For example, location (1.4, 12) indicates that the average of invalid received frames by Agent 1 is 1.4 and 12 for Agent 2.

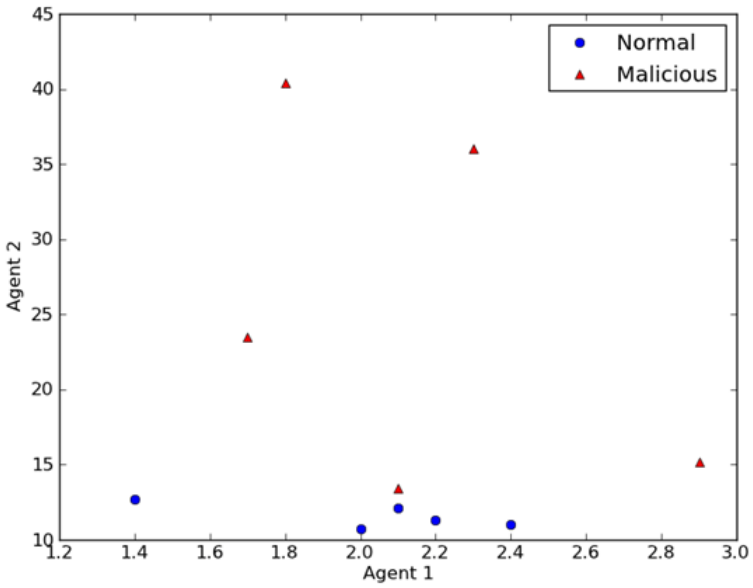

Figure 32: Agent 1 vs. Agent 2 mean values of received invalid frames

The interesting thing about this plot is that most of the normal activity is happening in one area. As Figure 32 shows the malicious activities are spread all over the area, which makes it suitable for the algorithm described by the flow diagram of Figure 33.

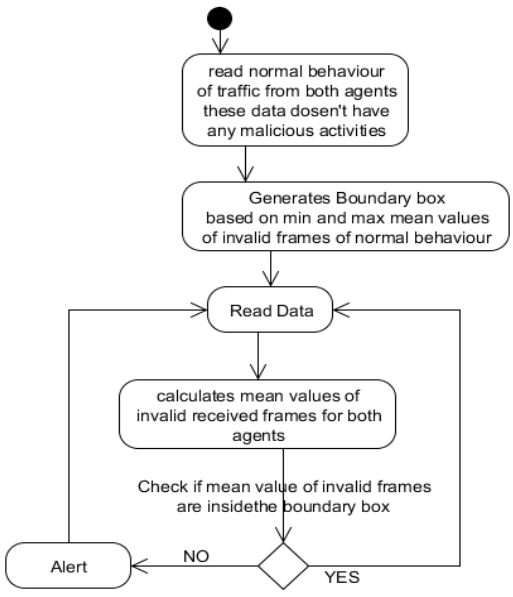

Figure 33: Multi Agent System Algorithm based on Boundary box

By using the above algorithm, which is shown in Figure 33, normal activity can be separated from malicious activity. The following figure, Figure 34, shows the demonstration of applying the algorithm for one of the experiment which has 2 agent nodes, 1 malicious node and 5 normal nodes.

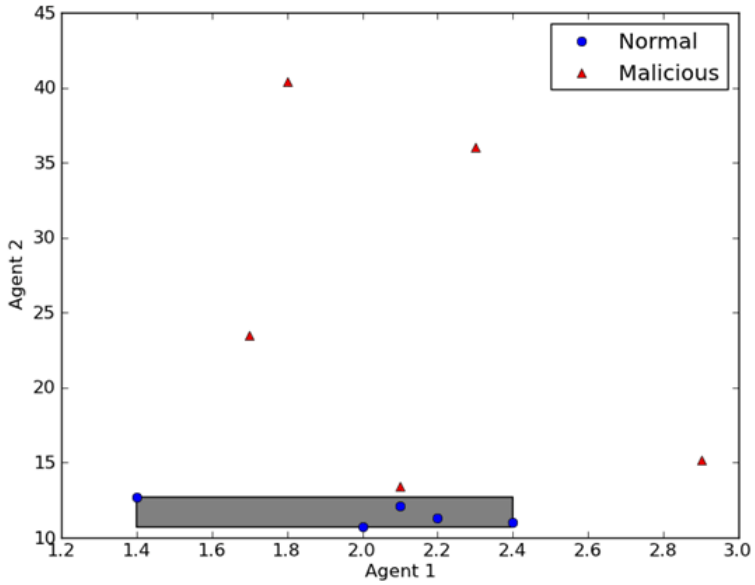

Figure 34: Agent 1 vs Agent 2 invalid mean value with applying Multi Agent System Algorithm

As illustrated by Figure 34, a boundary box is created which includes all the normal activities which are gathered during the first 50,000 rounds. Anything outside of this boundary box is flagged as malicious activity. In this case, those red triangles are flagged as malicious activities. This algorithm ran 32 times as described at the beginning of this section. The results were processed and Figure 35 and Figure 36 are generated for that.

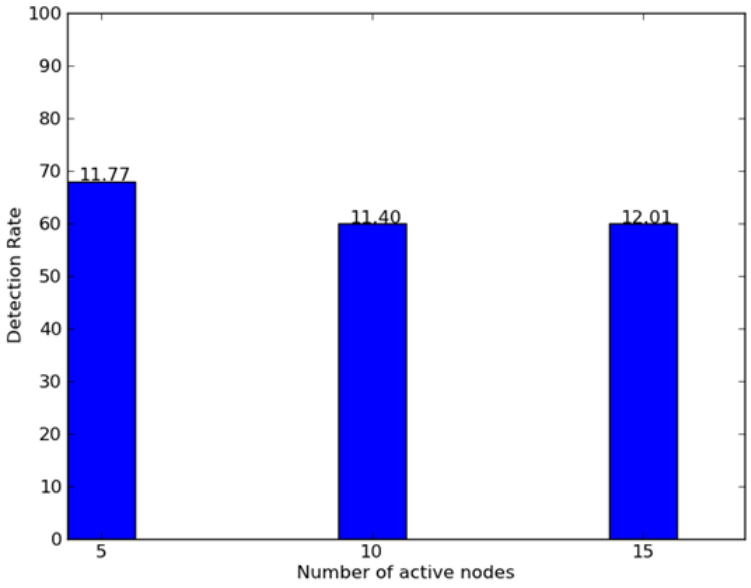

Figure 35: Detection rate vs. number of nodes for 2 agents and malicious node located in the green zone

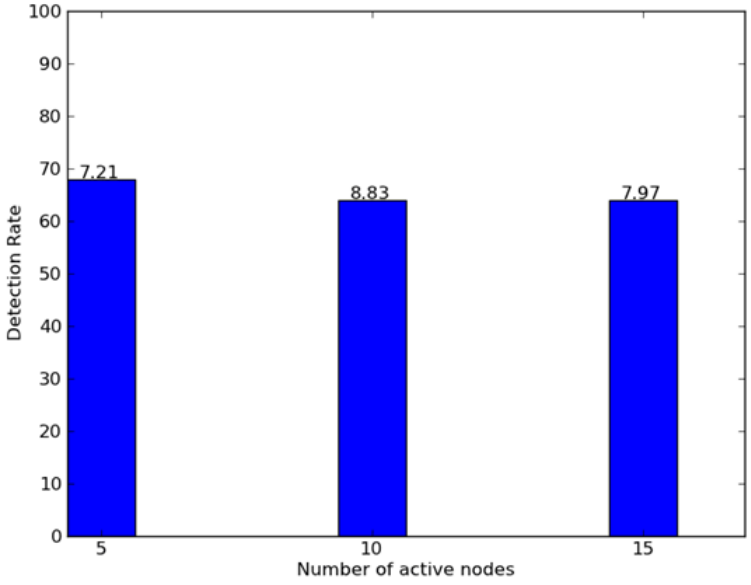

Figure 36: Detection rate vs. number of nodes for 2 agents and malicious node located in the red zone

Figure 35 and Figure 36 show that by using only this algorithm, almost 70% of covert-channel anomalies can be detected and flagged.

Chapter 6: Device implementation

During this thesis, the possibility of implementing the covert-channel communications using commercial devices both based on 802.11 and 802.15.4 explored. Fundamentally the challenge is to find a network card with a communication chip that allows for an application to use its own FCS algorithm. It was soon realized that the computation of the FCS is locked in the devices we analyzed and hard-wired into the communications chip set, not allowing these devices to send a frame with a different FCS than those available by the manufacturer. Some older communication chip sets are better suited for this than the newer ones. For example, Neufeld et al. [**25**] developed SotMAC – A Flexible Wireless Research Platform based on the DLINK DWL-AG530 PCI card that uses the Atheros AR5212 chipset that supported a flexible MAC layer. This network card and chipset are not manufactured any more. There are very few, if any, manufacturers that are developing a communication chip for 802.11 that support an application-based calculation of the FCS. Several promising devices have been tested such as Airpcap that support the capturing and transmission of nearly any form of packet, however, because of the inability to bypass the hardware computation of the FCS it is impossible to use this device as a test-bed.

Patrick et al. [**26**] investigated covert-channel communications over 802.15.4 devices like Telos-B, Telos-B is one of a few 802.15.4 communication devices that can support application computation of an FCS. The Telos-Bs utilized consists of the Texas Instruments CC2420 communication chip set that can support application computation of an FCS. To simplify development on the Telos-B the TinyOS environment is leveraged. A simple application was developed to communicate a short string using covert-channel communication to another

node. Unfortunately, after a certain number of frames have been sent and received over the coverts-channel, the wireless and serial communications began to fail due to complexity and some potential bugs in TinyOS. The mote which received the covert-channel frame also failed to receive any further frames. Because of the complexity of working with TinyOS, at what is effectively the driver layer, further work on developing a test-bed were postponed and the focus was placed on the use of a simulation environment.

Even though the Airpcap is not suitable for injecting frames into 802.11 wireless systems, it can be used as an agent and if it uses either Single Agent System or Multi Agent System approach, it can identify malicious nodes and detect malicious behaviour in 802.11 wireless networks.

Chapter 7: Conclusions

This thesis presented the establishment of a covert-channel in wireless networks in the form of frames with intentionally corrupted FCSs. Previous works had alluded to the possibility of using this kind of covert-channel as an attack vector.

In order to investigate detection of covert-channel communications, covert-channel had to be implemented into well-known simulators. Unfortunately, most of the simulators were not designed to allow purposely corrupted frames be injected into the communication stack. Sinalgo was chosen as a base simulator in this research and modifications have been applied and to force it to send and receive corrupted frames. Also a number of different node types have been discussed. For example, the agent node has been designed to monitor the network, and gathers statistical information. Normal node was designed to simulate the behaviour of normal activity in the simulation. At last, malicious nodes were introduced, which brought the covert-channel concept into the simulation.

Because there was no implementation of any kind for collision detection in Sinalgo, a uniform random distribution and a Poisson distribution model have been proposed as simple collision detection. They have been tested with a number of test case scenarios and in the end, Poisson has been chosen because it is known to resemble network traffic more accurately [35, ch. 29 p. 496].

In this research, statistical approaches have been investigated to see if they were suitable for detecting covert-channel communications. The only data that this research relies on was

number of invalid and valid frames per a given time. By using only these numbers this research tried to detect anomalies from large amounts of incoming data.

Single Agent System and Multi-Agents System were proposed. These two techniques have been designed to work based on history data. Historical data consist of a set of data collected during a period where the system exhibits normal behaviour, i.e., without the presence of covert-channel communication. In these two techniques multiple approaches such as Valid and Invalid data comparison and Boundary box techniques discussed in detail and some experiments were performed to show the advantages and disadvantages of both techniques. Also as the number of the simulation increases the randomness of the events is increases which make it ideal choice for testing against Single and Multi Agent(s) algorithms. The results of using both systems showed that almost 70% of anomalies, which in this case covert-channel, could be detected.

In Chapter 5, the possibility of implementing the covert-channel communications using commercial devices both based on 802.11 and 802.15.4 has been also explored.

The future work for this research can be number of things such as applying Single Agent and Multi Agent Systems into real hardware to see the detection rate of that. Also these systems can be improved to be used in mobile ad-Hoc Network to detect covert-channel in mobile scenarios. This research can be used as a base for improving the above algorithms and adding mobility into the scenario.

Bibliography

[1] Distributed Computing Group. (2007) Sinalgo, Simulator for Network Algorithms. [Online]. http://www.disco.ethz.ch/projects/sinalgo/ [Oct, 1, 2010]

[2] M. Barney. (2007) Modern Stegongraphy. [Online]. http://www.mikebarney.net/stego.html [Dec, 1, 2010]

[3] K. Szczypiorski, "HICCUPS: Hidden communication system for corrupted networks," in *The Tenth International Multi-Conference on Advanced Computer Systems ACS'2003*, Międzyzdroje, 2003, pp. 31-40.

[4] D. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," vol. 40, no. 9, pp. 1098-1101, September 1952.

[5] The Philosophical Research Society. (2001) Francis Bacon's Ciphers. [Online]. http://www.prs.org/gallery-bacon.htm [Dec, 1, 2010]

[6] V. Chu. (2009) ASCII Art Steganography. [Online]. http://pictureworthsthousandwords.appspot.com/ [Dec, 1, 2010]

[7] M. Odor, B. Nasri, P. Mason, M. Salmanian, M. Vargas Martin, R. Liscano, "A Frame Handler Module for a Side-Channel in Mobile Ad Hoc Networks," in *LCN Workshop on Security in Communications Networks (SICK)*, Zürich, 2009, pp. 930–936.

[8] W. Mazurczyk, K. Szczypiorski B. Jankowski, "Information Hiding Using Improper Frame Padding," in *14th International Telecommunications Network Strategy and Planning Symposium (Networks 2010)*, Warsaw, 2010, pp. 27-30.

[9] T. Sandford II, G. Theodore, "Hiding Data in the OSI Network Model," Weapon Design technology Group, Los Alamos National Laboratory, Los Alamos,.

[10] M. Wolf, "Covert Channels in LAN Protocols," in *Local Area Network Security (LANSEC)*, 1989, pp. 91–101.

[11] M. Fisk, C. Papadopoulos, J. Neil G. Fisk, "Eliminating Steganography in Internet Traffic with Active Wardens," in *5th International Workshop on Information Hiding, Lecture Notes in Computer Science: 2578*, 2002, pp. 18–35.

[12] G. Lewandowski, S.J. Chapin N.B. Lucena, "Covert Channels in IPv6," in *Privacy Enhancing Technologies (PET)*, 2005, pp. 147–66.

[13] D. Wagner, "Suggestions for the passing of passphrases," in *Usenet posting to sci.crypt and*

alt.privacy, 2005.

[14] S.J. Murdoch. (2007, December) Covert Channel Vulnerabilities in Anonymity Systems. PDF Document.

[15] R. Sbrusch. (2006) Network Covert Channels: Subversive Secrecy. Document.

[16] NS2. (1995-2011) NS2, The Network Simulator. [Online]. http://www.isi.edu/nsnam/ns/ [Oct, 1, 2010]

[17] OMNeT++. (2001-2009) OMNeT++. [Online]. http://www.omnetpp.org/ [Oct, 1, 2010]

[18] Scalable Networks. (2006-2011) QualNet. [Online]. http://www.scalable-networks.com/products/qualnet/ [Oct, 1, 2010]

[19] MathWorks - Matlab. (1994-2011) Simulink - Simulation and Model-Based Design. [Online]. http://www.mathworks.com/products/simulink/ [Oct, 1, 2010]

[20] Inc OPNET Technologies. (2011) OpNet. [Online]. http://www.opnet.com/ [Oct, 1, 2010]

[21] T. Kunz, E. McKnight-MacNeil, "Clock synchronization in WSN: Simulation vs. implementation," in *7th International Wireless Communications and Mobile Computing Conference (IWCMC 2011)*, Istanbul, Turkey, July 2011, pp. 980-985.

[22] D. E. Knuth, *The Art of Computer Programming, Semi numerical Algorithms*, 2nd ed.: Addison Wesley.

[23] G. V. Rossum. (2010) Python, Python Programming Language, version 2.7. [Online]. http://www.python.org [May, 1, 2011]

[24] MatPlotLib. (2008) matplotlib, python 2D plotting library. [Online]. http://matplotlib.sourceforge.net [May, 1, 2011]

[25] J. Fifield, C. Doerr, A. Sheth, D. Grunwald M. Neufeld, "SoftMAC – Flexible Wireless Research Platform," in *In HotNets-IV*, Maryland, 2005.

[26] M. Patrick, "Implementation of side-channel communication in Telus motes," Faculty of Businees and Information Technology, University of Ontario Institute of Technology, Oshawa, Internal Undergraduate Report 2010.

[27] X. Cao, Y. Li, R. Beyah Telvis E. Calhoun Jr, "An 802.11 MAC layer covert channel," *Wirel. Commun. Mob. Comput.*, 2010.

[28] Oracel System. (20011) JAVA, Object Oreintnted Programming Language. [Online]. http://www.java.com [Sep, 1, 2010]

[29] J. Anderson J. O. Arkin, "Ethernet frame padding information leakage," Atstake report 2003.

[30] A. F. Molisch, "Wireless Communications," John Wiley & Sons, ISBN: 047084888X , 2005.

[31] M. Asad, "Text Steganography Using Huffman Coding," in *ICIIT 2010*, 2010, pp. 445-447.

[32] G. C. Kessler. (2001, September) Steganography: Hiding Data Within Data. [Online]. http://www.garykessler.net/library/steganography.html [Dec, 1, 2010]

[33] J. Wingate. (2010, March) SARC - Digital Steganography Database Exceeds 800 Applications. [Online]. http://news.hostexploit.com/cyber-security-news/3562-digital-steganography-database-exceeds-800-applications.html [Dec, 1, 2010]

[34] K. Baras and A. Moreira, "Anomaly Detection in University Campus WiFi Zones," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on* , 2010, p. 202.

[35] R. Jain, *The Art of Computer Systems Performance Analysis*, 2nd ed. United States of America: John Wiley and Sons, Inc., 1991.