8-1-2007

Design and analysis of information fusion, dynamic sensor management rules for cyber security systems using simulation

Katie McConky

Rochester Institute of Technology

DESIGN AND ANALYSIS OF INFORMATION FUSION, DYNAMIC SENSOR MANAGEMENT RULES FOR CYBER SECURITY SYSTEMS USING SIMULATION

A Thesis

Submitted in partial fulfillment of the

requirements for the degree of

Master of Science in Industrial Engineering

in the

Department of Industrial & Systems Engineering

Kate Gleason College of Engineering

by

Katie Theresa McConky

B.S., Industrial Engineering, Rochester Institute of Technology, 2005

August, 2007

DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING

KATE GLEASON COLLEGE OF ENGINEERING

ROCHESTER INSTITUTE OF TECHNOLOGY

ROCHESTER, NEW YORK

CERTIFICATE OF APPROVAL

M.S. DEGREE THESIS

The M.S. Degree Thesis of Katie Theresa McConky

has been examined and approved by the

thesis committee as satisfactory for the

thesis requirement for the

Master of Science degree

Approved by:

_____

Dr. Michael E. Kuhl, Thesis Advisor

_____

Dr. Moises Sudit

ABSTRACT

Computer networks are vulnerable to attacks from outside threats. Intrusion detection systems are used to monitor computer networks for attacker activity. Intrusion detection systems consist of a set of sensors placed strategically throughout a computer network. The large amounts of data produced by intrusion detection system sensors may be sent to and processed by information fusion engines. Information fusion engines correlate alerts and identify attack paths of attackers. Sensor management strategies are developed to minimize the time taken to process attack data, minimize the bandwidth used by the security system of a network, and maximize the number of attacks successfully tracked. An experimental performance evaluation is conducted on sensor management strategies utilizing a variety of representative network topologies, network sizes, alert rates and attack scenarios so that a robust sensor management strategy can be identified. Performance measures of interest include the average time taken to process a real alert at the fusion engine, the percentage of real alerts processed, the percentage of noise alerts processed, the average bandwidth used to transfer alerts, and ability of a sensor management rule to successfully track multiple attacks consistently. Results indicate rules that attempt to meet but not exceed network constraints outperform rules that disregard network constraints. Additionally, rules that take into consideration the progress of current attacks also show some benefits.

TABLE OF CONTENTS

1. INTRODUCTION TO SENSOR MANAGEMENT IN THE CYBER DOMAIN

For several decades computers have been connected to one another in order to share access to information, programs and resources (Maran Graphics Inc., 1997). Networks were created to make computing more efficient and to reduce resource requirements. Networks allow multiple users to access the same file without the need of media transfer devices such as floppy discs, compact discs, or flash drives. Multiple users on the same network can access shared printers as well as licenses to expensive software.

Networks can range in size from a few computers connected in a Local Area Network in someone's home, to thousands of computers connected within a corporate office building. Networks can stretch between buildings and connect several buildings in a city or around the world. The largest and most well known network is the Internet which connects millions of computers worldwide (Maran Graphics Inc., 1997).

Networks that connect to the internet or some other external network are vulnerable to attack from enemies outside the network. Attacks from outside sources may include attempts to access secure information stored on servers, such as trying to access social security numbers, credit card numbers, or nuclear secrets held on file on secure servers. Attackers may also try to use a network's e-mail server to send unwanted and possibly malicious e-mail to thousands of recipients. Additionally, among many alternatives an attacker may create a denial of service attack, which floods the network with packets of unwanted information and creates so much traffic that legitimate network usage cannot take place (Crothers, 2003).

With the vulnerability of computer networks to attacks came ways to prevent attackers from gaining access to networks. Methods of preventing attackers from infiltrating networks are called intrusion prevention sytems because they are designed to prevent attackers from entering a

network. A well known intrusion prevention method is the use of passwords and user accounts to allow only people with access privileges to use the network. Other methods of intrusion prevention include firewalls which buffer a network's intranet from the external Internet by allowing access to the internal network only from trusted sources. While intrusion prevention devices work to limit access to a computer network, sometimes these devices fail to prevent all attacks, and an intruder succeeds in gaining access to the network (Crothers, 2003 and Bejtlich, 2004).

If an intruder successfully passes through a network's intrusion prevention system, there is another layer of network security that can be used to monitor an intruder's activities. This layer is called an Intrusion Detection System or IDS. An intrusion detection system monitors network traffic and host based activity for signs that an intruder is inside the network. Intrusion detection systems do not prevent network intrusions, but instead monitor the network to help identify if an intruder has passed through the intrusion prevention system and gained access to the network.

An intrusion detection system is made up of sensors placed strategically around a computer network. Sensors can be network based sensors or host based sensors. Network sensors, usually placed on switches, taps, and routers, monitor packets traveling between network machines, and often use pattern matching techniques to identify potentially harmful packets. Host based sensors are associated with a device such as a computer or server on the network, and monitor activities such as modifications to sensitive files or changes to security settings on an individual machine. The sensors follow predefined rules and generate alerts when an activity occurs that matches a certain format or signature (Crothers, 2003 and Bejtlich, 2004). For example, an alert may be generated each time a user enters his password incorrectly.

The sensor is not able to tell if the action that generated the alert is malicious or a result of ordinary network activity. The task of sifting through the alerts generated by IDS sensors to identify the alerts that correspond to malicious activity is often left up to a network analyst. Many alerts generated by the IDS sensors do not correspond to malicious activity. The alerts that do not correspond to malicious activity are often called false alarms, or noise. A false alarm is an alert that corresponds to a real network activity, but the activity was not intended to harm to the network (Crothers, 2003).

Intrusion detection sensors generate high volumes of alerts which are often sent to a central server for a network analyst to easily look at all the alerts. Alerts traveling to the central server use valuable network bandwidth to do so, and many of the alerts that are sent are not a result of malicious activity. The high volume of alerts arriving from a variety of sensors may easily overwhelm even the best network analyst (Crothers, 2003 and Sabata, 2006). Using an information fusion system can help to sort through the thousands of alerts in order to identify attacks, correlate alerts from multiple sensors, and track the path of the attacker through the network.

Due to the high number of alerts, the task of sorting through all the alerts to find the few malicious alerts and correlating the malicious alerts together is often impractical and difficult for a human analyst. Efforts have been made to automate the process using information fusion techniques. Information fusion engines use a variety of mathematical techniques to identify true alerts and find correlations between alerts to discover the attack path of the attacker (Chan, 2005, Sudit, 2005 and Sabata, 2006). In order for fusion engines to be effective they must receive information from the network sensors in a timely manner, so that an attacker's actions can be tracked in near real time. The ability of fusion engines to track an attack correctly may also be

influenced by the amount of noise in the dataset the engines receives. The less noise the fusion engines receive the faster the engines will be at identifying the attack track of the intruder. Figure 1-1 is a depiction of a simple network illustrating the use of intrusion prevention devices, network intrusion detection sensors, and two distributed information fusion engines.

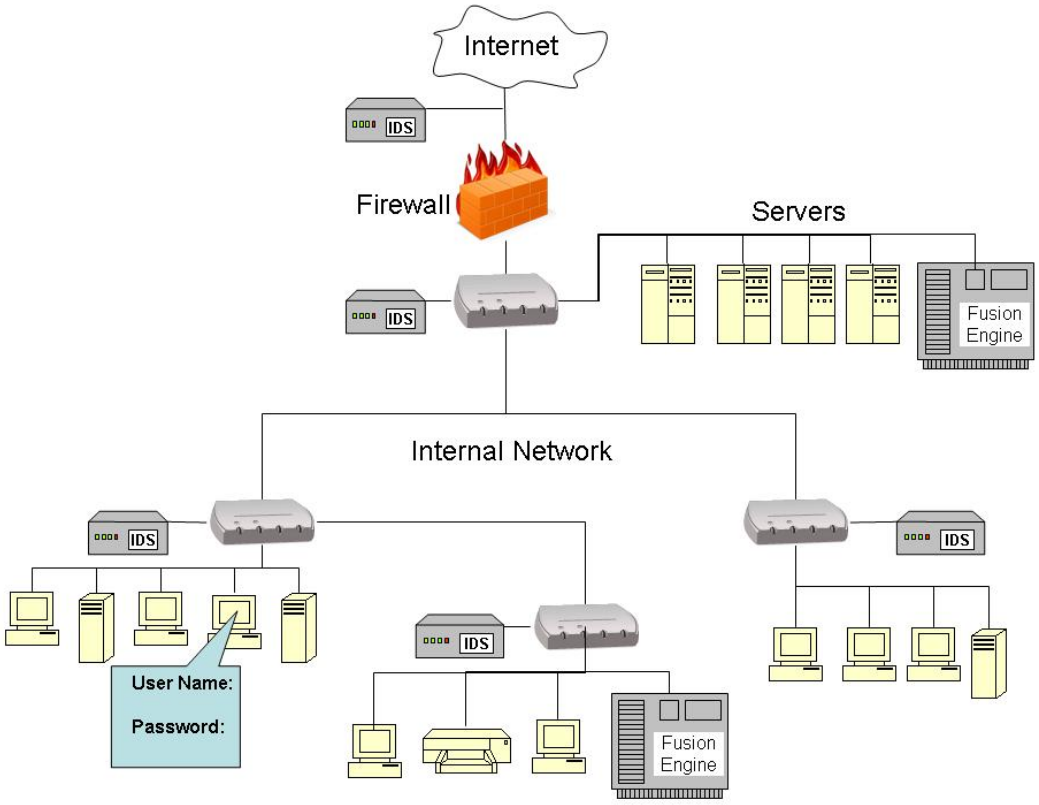

Figure 1-1 Network with Intrusion Prevention, Intrusion Detection, and Fusion Engines

In an ideal intrusion detection system, every alert generated by an intrusion detection sensor would be sent immediately to the fusion engine. At the fusion engine, the alert would be processed in a timely manner leading to the perfect tracking of all attack paths in the network. Unfortunately, there are a variety of constraints in a computer network that may limit the intrusion detection system. These limiting constraints include limits on the processing capabilities of the fusion engine and limits on the bandwidth available for system security. In

order to mitigate these constraints on fusion engine processing capabilities and available bandwidth, sensor management strategies can be employted.

This research looks at the interplay between the intrusion detection sensors and the fusion engines. Specifically, this research develops event based rules to direct data from the intrusion detection sensors to the fusion engines. These rules focus on sending the fusion engines the most pertinent information, in order to minimize the time required to process a real alert, minimize noise sent to the fusion engines, maximize the percentage of attack alerts that are processed, maximize the number of attacks tracked, and minimize network bandwidth usage.

2. PROBLEM STATEMENT

Sensor management strategies can have a significant affect on the performance of intrusion detection systems. The sensor management strategies developed in this thesis must be effective at minimizing the time to process real attack alerts, minimizing noise sent to the fusion engine, maximizing the number of attack alerts processed, maximizing the number of attacks tracked, and minimizing bandwidth used. A sensor management strategy that reaches all these goals will aid an information fusion system in effectively processing the most relevant IDS data to identify and take action against cyber attacks.

Several steps are taken to develop and test the sensor management rules. First, the rules themselves are designed for information fusion systems with processing and network bandwidth constraints. Second, performance measures are developed to use in the objective evaluation of the rules. Third, a cyber network simulation environment is created to test the developed rules. Finally, an experiment is designed and performed in order to identify the best performing rules as well as factors that significantly affect the performance of sensor management rules.

The main objectives of this thesis can be summarized as follows:

1. Develop event based dispatching rules that can be used to determine when sensors should send alerts to distributed fusion engines;
2. Develop metrics to track the effectiveness of the rules with respect to the time taken to process a real alert, the amount of noise processed at the fusion engine, the number of real alerts processed at the fusion engine, the number of attacks tracked, and the amount of bandwidth used to send alerts to the fusion engine;
3. Develop a simulation environment to model the sensor dispatching rules in the cyber domain;

4. Use the simulation to test the developed rules versus a variety of commonly used sensor management strategies;
5. Identify network factors that have a significant affect on the performance of sensor management strategies; and
6. Identify robust sensor management strategies for cyber security systems.

The goal of this research is to identify sensor management strategies that perform well for a variety of network configurations and attack scenarios, as well as provide strong results across a set of performance measures. A good sensor management strategy enables the fusion engines of an intrusion detection system to process real alerts in a timely manner. Fusion engines that process real alerts promptly, increase the effectiveness of intrusion detection systems by alerting analysts to attack tracks faster. The sooner an analyst learns of an attack in progress, the more chance an analyst will have to prevent the attacker from reaching his goal.

3. BACKGROUND ON NETWORK INTRUSION, INTRUSION DETECTION SYSTEMS, AND INFORMATION FUSION

. This section of the paper gives background information on attacker behavior, intrusion detection systems, and information fusion. First, a brief discussion on network intrusion and the anatomy of typical network attacks is discussed. This is followed by an extended discussion of intrusion detection systems so that they can be better understood in relation to this thesis topic. Finally, the intrusion detection discussion leads into a discussion of information fusion that is used to correlate intrusion detection sensor alerts.

Network Intrusion

While this paper is not a tutorial on network attack techniques, knowing that computer attacks often have several stages of an attack in common with one another is important. A computer attack can take any number of forms depending on the skill and the intent of the attacker. Attackers range in skill level from low skill level attackers, or script-kiddies, blindly running pre-coded scripts on networks to find well known entry points, to very highly skilled methodical attackers that take the time and energy necessary to plan out and implement a successful attack (Crothers, 2003).

The highly skilled attackers may have a variety of agendas for their attacks including, but not limited to, the following: data modification, data deletion, data access, system modification, program execution, privilege escalation, program installation, or security system detection avoidance(Crothers, 2003). To pursue any of these goals attackers commonly go through five phases of an attack: reconnaissance, exploitation, reinforcement, consolidation and finally pillage, as depicted in Figure 3-1 (Bejtlich, 2005).

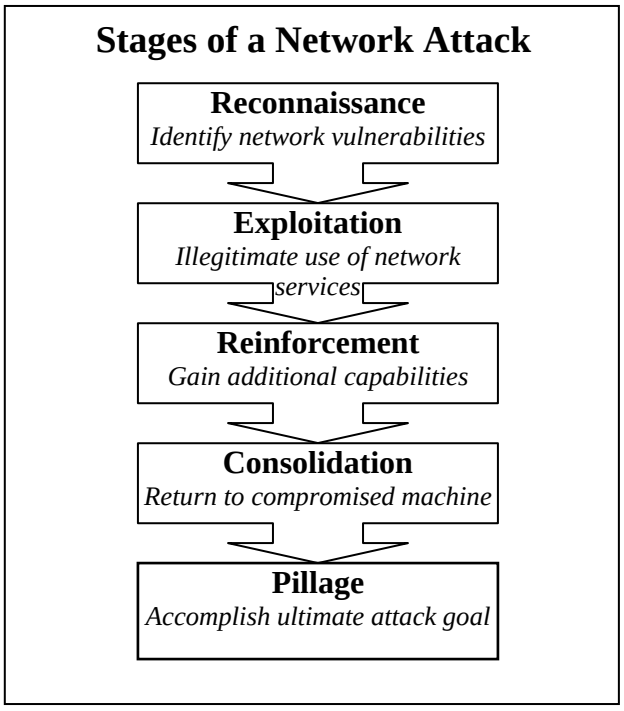

Figure 3-1: Stages of a Network Attack (Bejtlich, 2005)

The reconnaissance stage of the attack is the first phase in the attack sequence. During this stage the attacker attempts to identify vulnerabilities in the network security such as open ports or applications with known security problems. Reconnaissance is a very common step in structured attacks where the attacker has a goal for the overall attack, and wants to carry out the goal in the most efficient manner possible. Unstructured attacks, such as those run by script-kiddies, will often skip reconnaissance and run an attack on every network available to them in hopes an attack works on at least some. Reconnaissance can include non-technical activities too, such as digging in a corporation's trash for passwords and user accounts or paying insiders for information. These non-technical activities go undetected by intrusion detection systems (Bejtlich, 2005).

The second stage of the attack is called exploitation. During this stage attackers attempt to use a service for something other than the service's original design. The attackers may also make a service perform unusual tasks or stop a service from running altogether. If a service is

stopped, sometimes the attacker is able to use the privilege level of that service for his own use (Bejtlich, 2005).

The third attack stage is reinforcement. During this stage attackers gain additional network capabilities through their initial entrance into the network. They may choose to elevate their user privileges or transfer malicious code from their computer to the network under attack. During the reinforcement stage the attacker may create a backdoor to allow communication with computers outside the network, and to allow an easy reentrant point to the network that will go unnoticed by security (Bejtlich, 2005).

The fourth stage of the attack is called consolidation. During this stage the attacker returns to the network via the backdoor created earlier. Once he has gained access to the network he is free to do whatever he pleases as long as his privilege level allows the activity (Bejtlich, 2005).

Finally, the last stage of the attack is the pillage stage. During the pillage stage the attacker accomplishes her true goal of the attack. This may include stealing sensitive files, stealing user names and passwords, or setting up a system for a future attack (Bejtlich, 2005).

Knowing the attack steps that have already occurred is valuable when trying to identify the most pertinent information to send to a fusion engine. Unfortunately, these five attack steps are merely a guideline for attacker behavior, and are not necessarily followed by every attacker. Attacker behavior may be unpredictable because attackers are traveling blindly through a network, and do not necessarily know the location of their target machine. Additionally, attackers are constantly finding new and surprising methods to infiltrate networks that may go unnoticed by intrusion detection systems (Holsopple, 2006).

Intrusion Detection Systems

As described in the introduction most networks have a layer of security including firewalls that aims at preventing intruders from entering the network. These systems are invaluable for protecting network security, but due to the craftiness of modern attackers they can often be compromised and fail. The failure rates of intrusion prevention devices prompted the development of intrusion detection systems in the late 1990's as a second blanket of security (Bejtlich, 2005). Intrusion detection systems do not prevent attackers from entering the network, but act as an alarm system that alerts security to an attacker's potential presence.

Intrusion detection systems (IDS) for computer networks are made up of a set of sensors that are placed strategically around a network to look for suspicious activity. When suspicious activity is found, sensors generate and send an alert to the security analyst, often times a single person, to notify him of the activity. Intrusion detection systems work in a variety of methods to identify potential threats. The techniques sensors use to identify threats differ depending on the location of the sensor and the approach employed by the system as a whole.

Intrusion detection sensors are either network based sensors or host based sensors. A network based sensor is often associated with a router, switch, hub, or firewall, and its job is to look at all the packets of information traveling through the device. When the network sensor scans each packet the sensor performs pattern matching tasks to match the contents of the packets to known attack signatures. Pattern matching may be done on a single packet or on multiple packets to find malicious code that spans several packets (Crothers, 2003). If the network has a very high bandwidth, or a particularly busy switch, multiple sensors may be used to ensure that every packet is scanned. Network based sensors merely sense whether a packet matches an attack signature or not, the sensors do nothing to stop a packet if a malicious

signature is found. Some advantages of network based sensors are that network sensors detect both successful and unsuccessful attacks, network sensors are separate from host machines so they are less vulnerable to be tampered with by intruders, and network sensors can detect a large variety of malicious activity (Crothers, 2003).

Accompanying network sensors to monitor the network are host based sensors. Host based intrusion detection sensors are sensors that monitor activities on a host such as a server or a computer workstation. These sensors create logs to track suspicious activities on machines, such as tracking changes made to secure files. Host based sensors are implemented using software installed on each machine. The host based sensors use processing resources of the host, so setting security too tight will slow down the machine processing. Compared to network sensors, host based sensors are better at detecting new attacks and they create fewer false alarms than network sensors. Unlike network sensors, intruders may have the ability to modify sensor logs on a compromised machine if the host based sensors are not adequately secured (Crothers, 2003).

A good intrusion detection system has both network and host based sensors in order to successfully track all manner of malicious activity. These intrusion detection sensors often times generate alerts for activities which are not malicious. For example, an alert may be generated when a user mistypes their password when trying to access the network. Intrusion detection systems are known to have high false alarm rates, and generate many false positives. A false positive is an alert that is genuine, but does not correspond to malicious behavior. A false negative occurs when no alert is generated, but an attack has occurred. False negatives may occur for new attacks or if an attacker is particularly stealthy. The rate of false positives to actual malicious alerts can be as high at one hundred to one (Crothers, 2003), especially when intrusion

detection systems are first put in place. The rate of false positives will be referred to as the noise rate in this thesis.

Intrusion detection systems can be purchased as a security package available from a variety of retailers, but many systems and software packages are available for free such as Snort, Dragon, and Daiwatch (Bejtlich, 2005). Intrusion detection systems must have a security analyst (or several) available to sort through all the alerts generated during a typical day. Data from sensors is often sent to a secure intrusion detection server for a human analyst to look at. Efforts are being made to make the human analyst's job easier by automating the attack detection process using information fusion techniques. Information fusion engines sort through all the alerts generated by the intrusion detection sensors, and through correlation and other techniques are able to identify and track attacks and eliminate the noise.

Information Fusion

Due to the large amount of disjoint information being retrieved from the intrusion detection sensors, analysts are often overwhelmed by the amount of data they must wade through in order to identify a true attack. A group of well trained sensors watching 64,000 addresses can generate 25,000 to 50,000 alerts/day and over 100,000 alerts/day during peak activity (Sabata, 2006). Information fusion is a natural solution to the problem of large amounts of disjoint data. A well designed information fusion engine can aid the human analyst to better understand all the alerts that are generated. As defined by the Australian Department of Defense, information fusion is "A multilevel, multi-faceted process dealing with the automatic detection, association, correlation, estimation, and combination of data and information from single and multiple sources" (Kang, 2003). With respect to network intrusion detection this means an information

fusion engine identifies alerts that correspond to real attacks, correlates them with alerts from other sensors to find the attack path of the intruder, and eventually predicts what the intruder may do next.

According to the Joint Director's Laboratory there are five defined levels of information fusion, levels zero through four. The thrust of this thesis deals with levels zero and four. A flow chart of intrusion detection fusion can be seen in Figure 3-2, followed by a brief discussion of each fusion level.

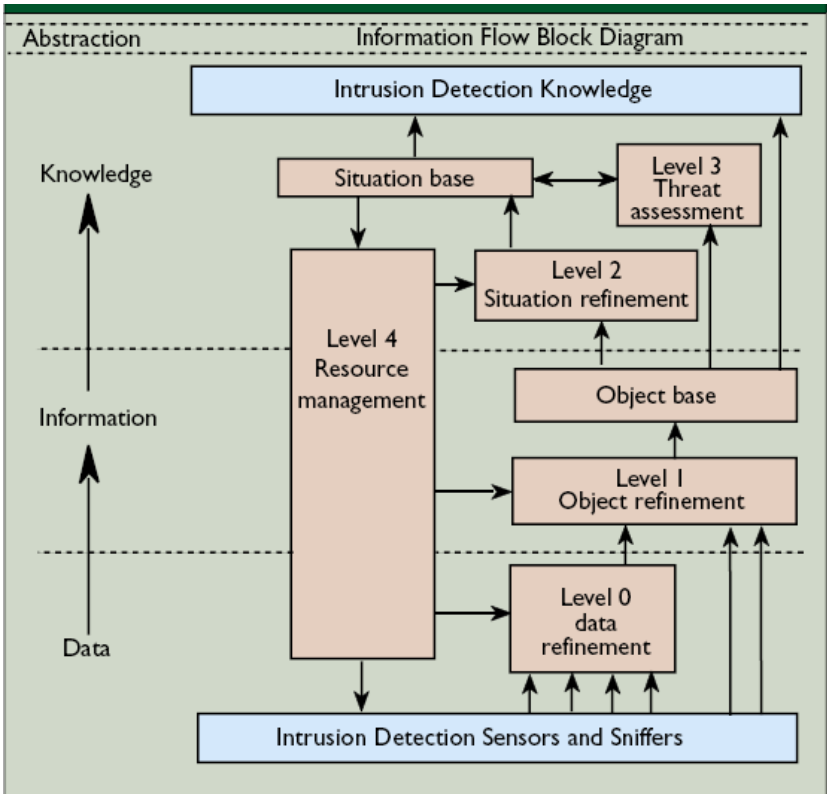

Figure 3-2: Intrusion Detection Data Fusion (Bass, 2000, p.102)

Fusion level zero is characterized by sub-object data assessment, or data refinement. With respect to intrusion detection this corresponds to the alerts generated by the IDS sensors. Level one fusion is object refinement. In cyber security this corresponds to relating the alerts generated by the IDS sensors to certain kinds of attacks. Level two is situation assessment or situation

refinement which looks at objects and their relationships to assess the current situation. In cyber security, fusion level two relates attack alerts to other attack alerts to determine the path of the attacker. Level three is impact assessment or threat assessment. This corresponds to determining what the target of the attack might be, and what will be the effect if the attack is successful. Level four fusion is process refinement and often deals with resource management (Sudit,2005). This paper focuses on process refinement with respect to retrieving data from intrusion detection sensors using a variety of dispatching rules.

Many information fusion platforms have been put in place to use information fusion to help identify and track attacks through a computer network (Sabata, 2006, Sudit and Holsopple, 2006). These fusion engines can correlate data from multiple sensors in order to find attacks that may be related, track attacks through the network, and eventually predict what the attack target is. Regardless of fusion engine used, they require data, and most fusion engines are currently requesting all the alerts from every intrusion detection sensors.

4. LITERATURE REVIEW

In many sensor applications, intrusion detection systems included, the sensors gather a vast amount of data. Most cyber security sensor systems assume that all the data gathered by the sensors can be sent to the fusion engine, and furthermore that the fusion engine is capable of processing all of the data acquired in a timely manner. The ability to send all data to the fusion engine is not a valid assumption because there are many constraints in the network (Wang, 2004), likewise the ability of the fusion engine to process all sensor data may be invalid because of constraints in the processing power of the fusion engine. Due to network bandwidth constraints and fusion engine processing constraints, only a subset of data generated by all the sensors can be sent to the fusion engine. Which sensors to pull data from, at what time, and what data to pull from the chosen sensors become important questions to ask in a resource constrained network.

Bandwidth constraints exist in computer networks. Even though standard computer networks often have high bandwidth, sending too many sensor messages has the potential to slow down the communication for daily network activities and cause slow network performance. While there may not be a tight constraint, minimizing bandwidth use by network sensors is important in order to not interfere with required network performance.

There has been a significant amount of research recently in the field of sensor management and sensor scheduling. Much of the research is applied to distributed sensor networks with tight bandwidth and energy constraints. While cyber networks do not have the same tight bandwidth constraints, much can still be learned from the research. Research is focused mainly on using algorithms to schedule sensors, using weights for information quality,

and applying novel concepts such as network calculus or market based management to schedule sensors.

There are many examples of research that use algorithms and integer and linear programming to schedule sensors. Algorithms offer a promising solution to sensor management because when executed these algorithms can provide an optimal sensor schedule. Algorithms have been developed to schedule the polling of data from sensors in order to maximize information value of data gathered while not exceeding bandwidth and energy constraints of the network. Unfortunately, often when using algorithm scheduling techniques, the algorithms developed are too difficult to solve in a timely manner. Heuristics, such as a one step ahead method or other techniques, are used to provide sub-optimal solutions to scheduling algorithms in real time (Sciacca, 2002 and Krishnamurthy, 2002).

Another drawback of much of the algorithm based research is that the focus is only on polling one sensor at a time due to the assumption of a heavily bandwidth constrained network. In the case of network intrusion detection sensors, bandwidth is not as severe a constraint. More than one sensor can be polled at a time in a computer network, and in the cyber realm the focus is more on minimizing bandwidth use rather than meeting an extreme bandwidth constraint. These algorithms also assume that sensors are controlled by a central sensor polling agent, and do not allow sensors to decide for themselves whether they should send data or not.

An interesting approach in use to enhance the ability of algorithms to properly schedule data collection is to split up a single sensor into multiple “virtual sensors”. Each virtual sensor holds a different type of information or a different quality level of information. For example, for a temperature sensor, one virtual sensor may hold temperature measured to the nearest degree, while another virtual sensor may hold the temperature to the nearest hundredth degree, while yet

another virtual sensor may only hold the binary information of whether the temperature is above freezing or not. By using virtual sensors, data can be polled that fits best with the current bandwidth constraints (Sciacca, 2002).

The flexibility of an algorithm approach allows a variety of considerations to be incorporated into the models. One algorithm includes the recognition that some sensor measurements may take more time and contribute more value to the overall observation than other sensors, so that sensors should not all be treated equally. The use of Hidden Markov models can make the past observations and past sensor choices influence the sensor that is chosen next (Krishnamurthy, 2002).

Some algorithm based research is focused heavily on giving weights to sensors based on data quality (Zhang, 2002 and Nicholson, 2004). Weights are been used to assign values for data quality, data timeliness, and data importance related to the overall sensor network objective. The weights are used to effectively choose the data that would best serve the network objective (Zhang, 2002). Weighted values are also used to differentiate between the quality of the information and the information value. Information value is defined as a measure of relevance related to "how much does the information cost to acquire and what is the expected pay-off as a consequence of action upon it" (Nicholson, 2004, p.129).

Assigning weighted values to sensor data addresses the problem of a sensor that produces so much information that only a subset can be sent to the fusion engine. The results of the scheduling system, which use a one step ahead scheduling algorithm, are compared to a sensor scheduler that went in cyclical order giving each sensor a chance to send data. The scheduler that took information value and quality into account naturally performs better (Nicholson, 2004).

Other approaches to sensor management include a network calculus approach that uses network calculus to mathematically find resource allocation so as not to exceed bandwidth (Zhang, 2002). The network calculus approach focuses on a dynamic environment where one might want to look at different sensors depending on the current picture of the environment. The research focuses on a hierarchal distributed sensor network that was not very similar to the cyber intrusion detection sensor network.

Another novel approach to sensor management uses a market based approach to effectively sell sensor resources to various buyers (Mullen, 2006 and Viswanath, 2005). The approach uses combinatorial auctions so that buyers can bid on the services of several sensor combined. The market based sensor management approach seems more aimed at a sensor network that is providing information to several fusion engines that are competing for the same sensor resources, but the market based approach is interesting and unique.

Research is done in the area of using sensor networks to track mobile targets, such as airplanes, traveling over a set of distributed sensors. In such studies mobile targets are assumed to originate beyond the borders of the sensor network. The sensors' job is then to track the mobile target as the target passes through the space monitored by the sensors. One approach taken to schedule such a sensor network is to keep all sensors on the perimeter of the sensor network on and sensing at all times. Keeping the perimeter sensors on, allows the network to consistently observe new targets entering the sensor area. While the perimeter sensors remain on at all times, the interior sensors are put into hibernation mode. Once a target is observed by the perimeter sensors, specific internal sensors are turned on based on a predicted path of the target (Yang & Sikdar, 2005). This approach works well at effectively tracking targets while conserving the limited sensor energy.

Some aspects of mobile target tracking are similar to tracking attacks in a computer network. In both situations the target is assumed to come from outside the network. In the case of a mobile target, the mobile target is assumed to approach the sensor network from outside the network's boundaries. In the cyber network case, an attacker often enters the network through computers with access to an external network. The sensors on the computers with external access are analogous to the sensors on the perimeter of a distributed sensor network set up for tracking mobile targets. An approach similar to the target tracking approach could be used in a cyber environment to track cyber attacks. By requiring sensors on computers with external access to constantly send their alerts to the fusion engine, the fusion engine will be able to consistently observe new attacks as they start in a network. Additionally, only turning on sensors on machines that are vulnerable to attack based on the attack's current path can also prevent unnecessary noise alerts from being sent to the fusion engine.

While research is done in the area of sensor scheduling and sensor management, many of the papers are concerned with wireless distributed networks with extreme bandwidth constraints. Most papers research deals with meeting extreme bandwidth constraints rather than minimizing bandwidth use to have less impact on the network. The research often deals with energy and bandwidth constraints in the network, rather than processing constraints of the entity processing the sensor information. The papers often present complicated algorithms that are too difficult to solve in a timely manner, and none were found that presented simple queuing strategies or event based rules to schedule sending sensor data.

Future research can use some of the details of the existing sensor management research such as including the value of data held at the sensors in scheduling rules, developing virtual sensors to store different types of alerts, and comparing results to cyclical or random rules. This

paper attempts to develop event based rules that try to reduce bandwidth use and reduce noise sent to the fusion engine in order to better track network attacks while reducing use of network resources. Additionally, this research develops rules that are easy to implement and do not involve complicated intractable algorithms.

5. SENSOR MANAGEMENT RULES

Sensor management rules are used to mitigate the constraints of intrusion detection systems, including fusion engine processing constraints and bandwidth constraints. Sending all alerts generated to the fusion engines may not be a practical management strategy because fusion engines have limited processing capabilities. In order for information fusion engines to remain responsive to current network condition, sensor management strategies must closely monitor the use of the fusion engine resource. Additionally, sensor management strategies must ensure that bandwidth usage does not exceed the amount set aside for security use.

There are four resource situations that an intrusion detection system may encounter while attempting to process alerts: stable, under fusion engine capacity, over fusion engine capacity, or over bandwidth capacity. When the intrusion detection system is stable, the number of alerts sent to the fusion engine does not exceed the bandwidth allocated to the security system, while at the same time enough alerts are delivered to the fusion engine to keep the fusion engine busy. When the system is sending all desired alerts to the fusion engine, but the fusion engine has extra capacity available to process alerts, the system is said to be in the under fusion engine capacity state. If the number of alerts sent to the fusion engine exceeds the capacity of the fusion engine to process the alerts and the queue at the fusion engine has grown above a predefined threshold, the system is in the over fusion engine capacity state. Finally, when the number of alerts sent to the fusion engine exceeds the bandwidth allocated to the security system, the system is in the over bandwidth capacity state.

To accommodate the four possible states that the system could be in, a tiered rule structure was developed. The rule structure specifies a main rule that is used to identify the ‘wants’ of the fusion engine, then, depending on the state of the system, a subset of the fusion

wants, all of the fusion wants, or all of the fusion wants plus additional alerts will be sent to the fusion engine. The fusion engine 'wants' are defined as the alerts that the fusion engine would most like to receive if the system had no constraints. The diagram below in Figure 5-1 shows the rules available in each state, each of which is described in the following sections.

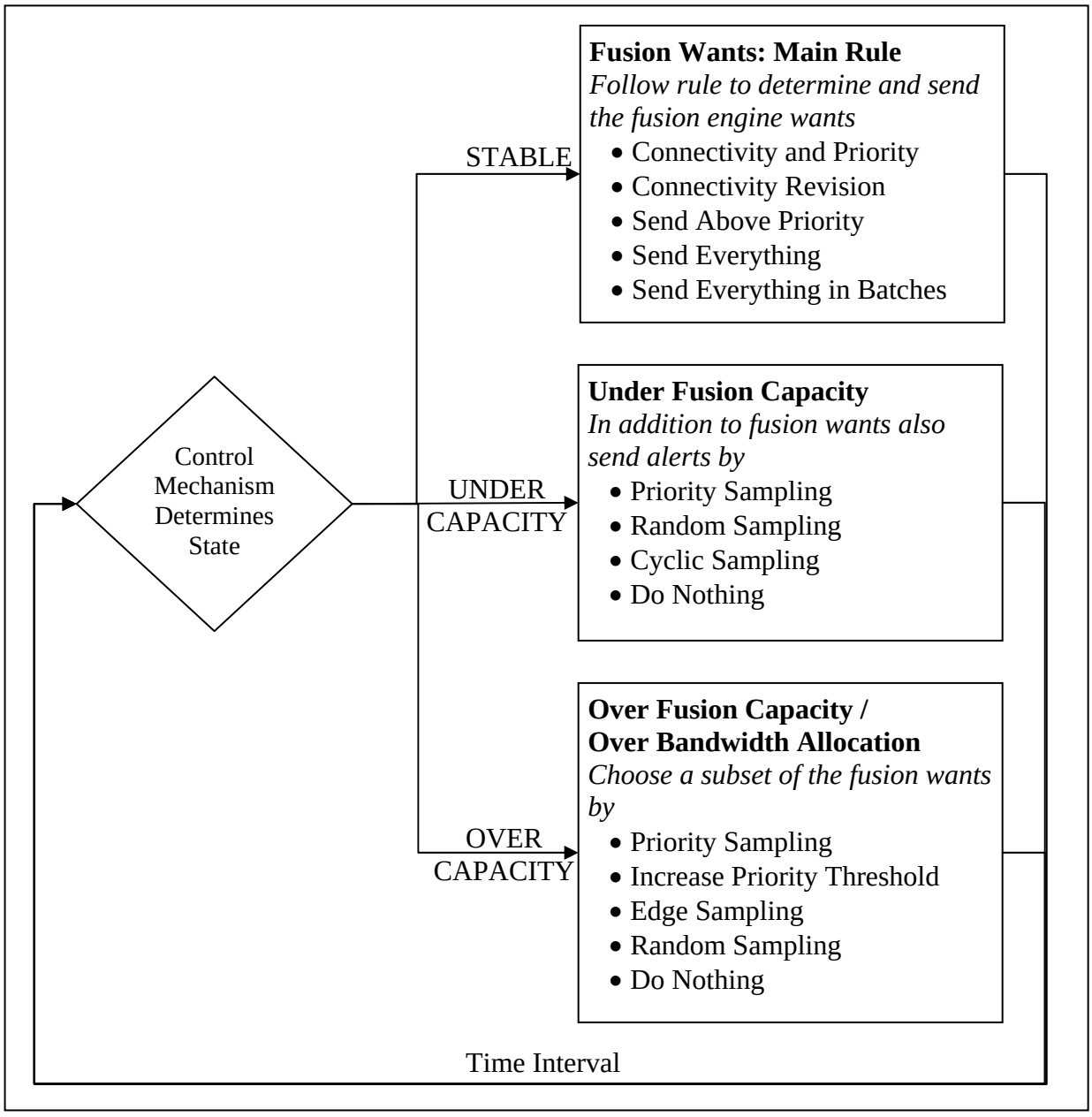

Figure 5-1: Rule Hierarchy

Monitoring System State

In order to monitor the state of the system, a check is performed at regular time intervals, in the case of this experiment an interval of one second is used. During this check, the amount of bandwidth used for that second, the number of alerts sent to the fusion engine that second, and the length of the fusion engine queue are checked. The amount of bandwidth used, the number of alerts sent to the fusion engine, and the length of the fusion engine queue combine to determine whether the system is in the stable, under capacity or over capacity states.

The sensor management rules start by following the rule selected for the stable state, referred to as the main rule of the sensor management strategy. The rule followed during the stable state determines the fusion engine wants, or the set of alerts that the sensor management strategy would send to the fusion engine if the system were unconstrained. The stable system may have a small queue built up at the fusion engine, but the queue at the fusion engine has not yet reached the critical length required to activate the over capacity state. Additionally, the bandwidth used for a system in stable state is not exceeding the bandwidth allotted to the security system. Figure 5-2 depicts a system in stable state.

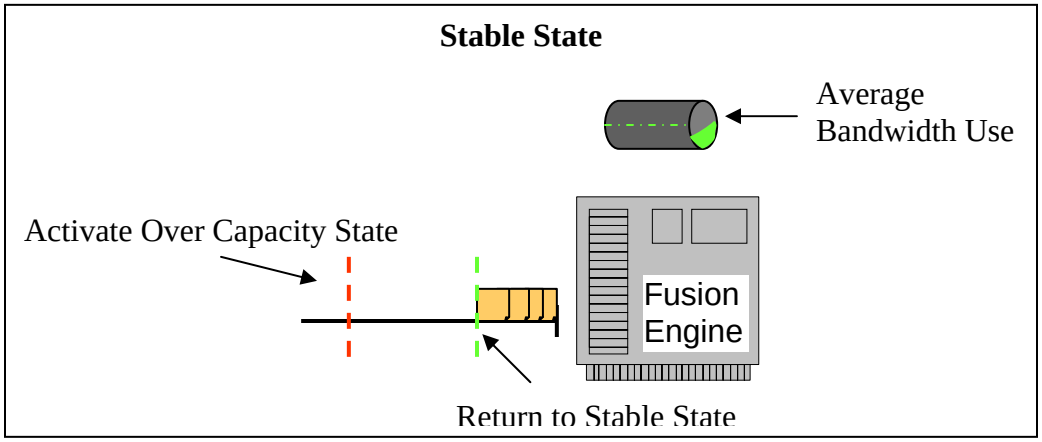

Figure 5-2: Stable System State

A system enters the over capacity state if at least one of the following two conditions is met: the length of the fusion engine queue exceeds a predefined critical length or the number of alerts waiting for bandwidth availability exceeds the same predefined length. Once in the over capacity state the system follows the pre-selected over capacity rule. The over capacity rule chooses a subset of the fusion engine wants to send to the fusion engine. The system remains in the over capacity state until the queue at the fusion engine or the queue of alerts waiting for bandwidth recedes to a predefined minimum threshold level. Once the offending queue recedes to the minimum threshold the system once again enters the stable state. Figure 5-3 depicts a system in the over capacity state caused by both a long fusion engine queue and high bandwidth usage.

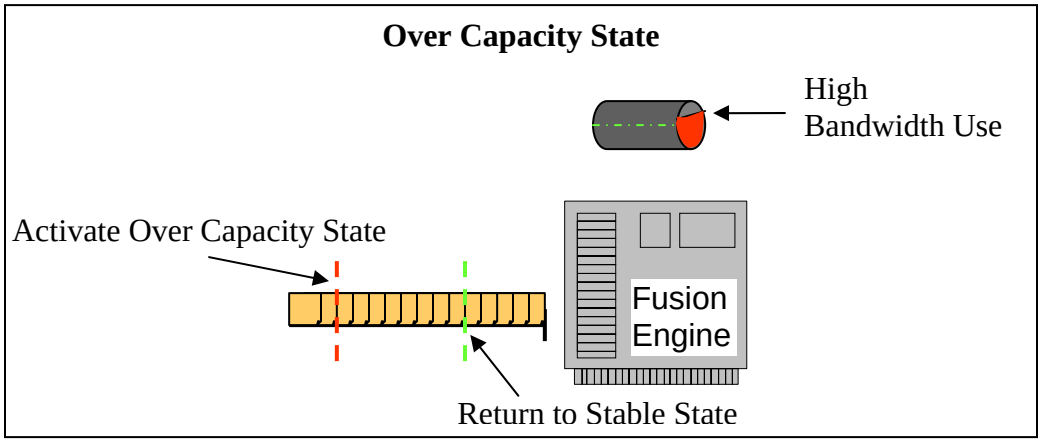

Figure 5-3: Over Capacity System State

The system may be in the under capacity state only if the system is not in the over capacity state. If the system is not currently in the over capacity state, and the number of alerts sent to the fusion engine was less than the number of alerts the engine can process in a second, the system goes into the under fusion capacity state. The system then follows the designated under fusion capacity rule in order to sample an amount of alerts equal to the remaining fusion engine capacity for that second. The under fusion capacity state does not affect the main rule, the

state merely sends additional alerts to the fusion engine above the defined fusion wants. A system in the under capacity state would have a very short or nonexistent fusion engine queue and very low bandwidth usage as depicted in Figure 5-4.

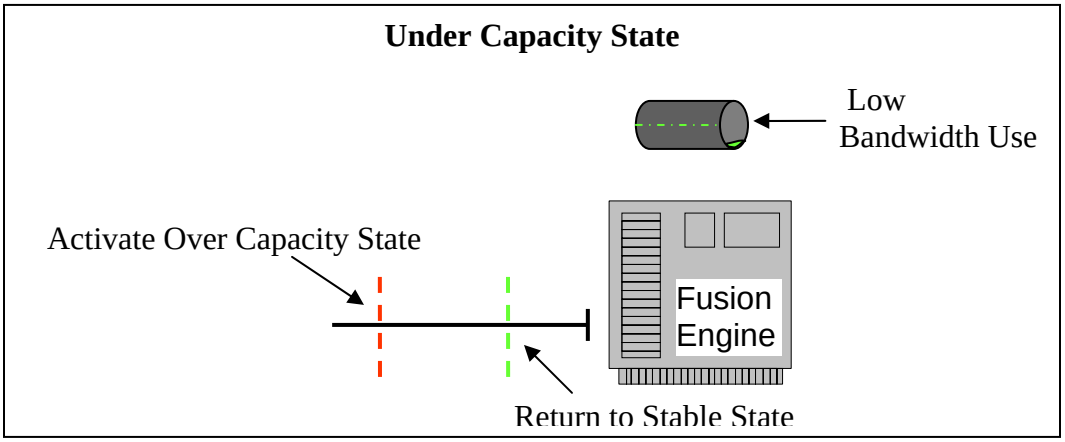

Figure 5-4: Under Capacity State

Determining Fusion Engine Wants

The fusion engine wants rule is the main rule of the intrusion detection system. This rule specifies which sensors should send alerts to the fusion engine, and which alerts from these sensors should be sent. The rules available to choose from include the Connectivity and Priority Rule, Connectivity Revision, Send Above Priority Rule, Send Everything Rule, and Send Everything in Batches Rule.

5.2.1 Connectivity and Priority Rule

The Connectivity and Priority rule uses the progress of an attack through a computer network as well as network topology to determine the settings of the sensors. The Connectivity and Priority rule sets a sensor to follow one of three sensor rules: External Monitor Rule, Internal Monitor Rule, or Standby Rule. The external monitor rule has the highest priority over all the

rules, followed by the internal monitor rule, and then the standby rule. The priority of a rule is used when resetting sensors after attacks have been completed.

When starting a new scenario only the sensors with external access are allowed to send alerts to the fusion engine, all other sensors are set to standby. A sensor in standby mode receives and stores all alerts, but does not attempt to send any alerts to the fusion engine. The initial sensor settings are recorded in a default attack set that maintains the settings of the sensors before any attacks have progressed through the network.

The external monitor rule has a specified minimum priority and maximum batch size. The minimum priority corresponds to the minimum priority required of an alert before the alert can be sent to the fusion engine. The maximum batch size is the largest number of alerts allowed to be stored at a sensor before they are automatically sent to the fusion engine. Sensors set to the external monitor mode may send an arriving alert to the fusion engine only if the alert is above the specified minimum priority. If an alert arrives at an external monitor sensor that is not above the minimum priority then the alert is stored at the sensor until the maximum batch size is reached or the rule requests the alert. Once the maximum batch size is reached, all alerts batched at that sensor are sent to the fusion engine. A flowchart of the actions taken by a sensor operating under the external monitor rule can be seen in Figure 5-5.

As the fusion engine processes the alerts received, attack tracks are created for attacks in progress. An attack track is a series of machines that have been targeted in an attack. The fusion engine will generate a new attack track for each unique attack in the attack scenario. When an attack track is created, altered, or a new alert associated with an existing attack track is found, the fusion engine notifies the Connectivity and Priority rule.

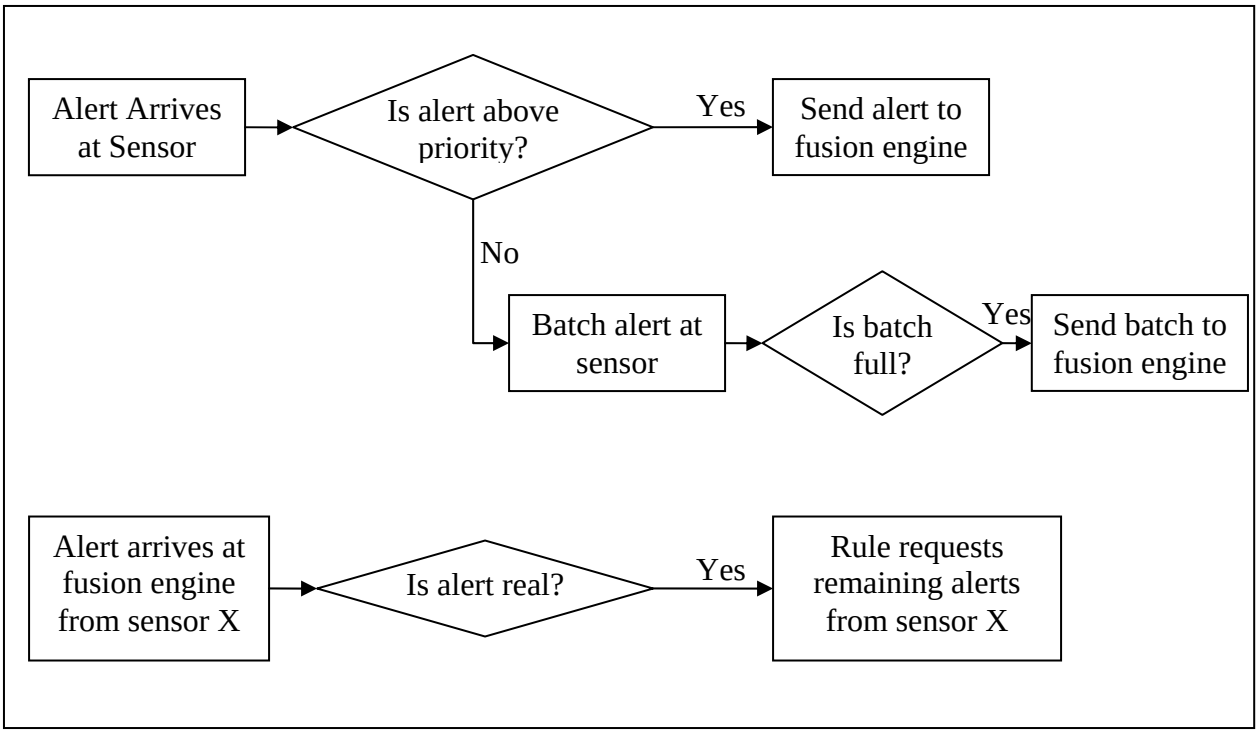

Figure 5-5: Connectivity and Priority: External and Internal Monitor Rule Flowchart

When the Connectivity and Priority rule receives a change made to an attack track, the rule ensures that all sensors accessible from any of the attacked machines are set to the internal monitor mode. Sensors not connected to any machine involved in the attack continue to remain in standby mode. When the fusion engine informs the Connectivity and Priority rule that a real attack alert was received from a sensor, the rule asks the sensor to send all of the alerts in the sensor's batch to the fusion engine.

When a sensor is initially set to internal monitor mode any alerts received at that sensor after the time of the alert that caused the sensor to be switched to internal monitor mode are sent immediately to the fusion engine. All other older alerts are archived, essentially removed from the sensor alert queue, and are never sent to the fusion engine. The process of switching from the standby rule to the internal monitor rule is illustrated in Figure 5-6. The internal monitor mode sensor acts similarly to a sensor in external monitor mode. As alerts are received at a

sensor in internal monitor mode, the alert is checked to see if the priority is greater than or equal to the minimum priority level. If the received alert is at or above the minimum priority level the alert is sent immediately to the fusion engine. All other alerts are batched at the sensor. Alerts not sent to the fusion engine are placed in a queue based on priority and time, with the highest priority and most recent alert placed first in the queue and the oldest alert with lowest priority placed last in the queue.

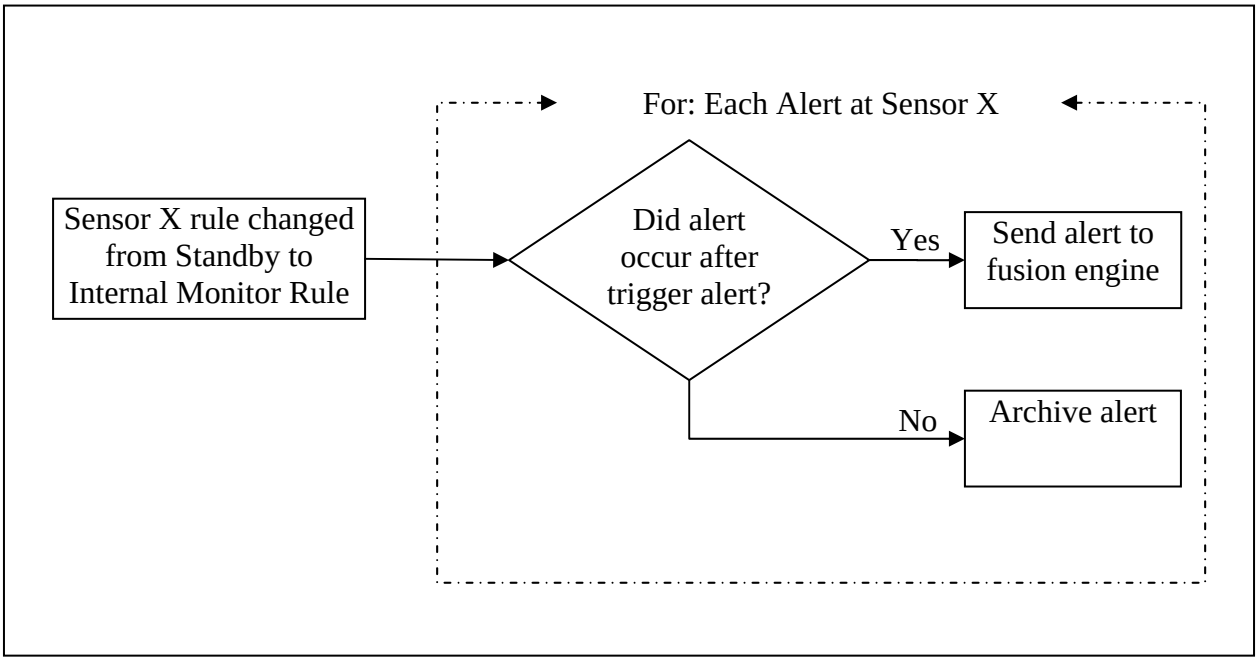

Figure 5-6: Connectivity and Priority: Switching from Standby to Internal Monitor Rule Flowchart

Alerts are only removed from the batched queued alerts if a real alert from that sensor is processed by the fusion engine. Once a real alert from a sensor is processed by the fusion engine, the Connectivity and Priority rule is notified, and the remaining alerts in the batch are requested from the sensor.

As an attack progresses down through a network, more and more sensors are switched from standby to internal monitor mode. As each attack progresses, the Connectivity and Priority rule creates an attack set for each attack. Each attack set contains a list of sensors that are in each

of the three sensor modes: external monitor mode, internal monitor mode, and standby as determined for that attack. As attacks are complete the attack sets generated by the Connectivity and Priority rule can be removed from the system and sensors can be reset as necessary.

An attack is considered complete only if the attack scenario is done creating alerts for that attack, all sensors no longer contain alerts from that attack in their queues, and the fusion engine queue does not contain any alerts from that attack. Once an attack is complete, the fusion engine notifies the Connectivity and Priority rule. The attack set associated with the completed attack is then removed from the set of active attack sets. All sensors are then reset to the highest priority rule that all other attack sets have specified for each sensor. If a sensor is not involved in any other attack sets, the sensor is returned to the default setting defined at the beginning of the scenario run.

5.2.2 Send Above Priority Rule

Another rule available to determine the fusion engine wants is the Send Above Priority Rule. The Send Above Priority rule allows the user to specify a minimum priority level. The sensors then send all alerts at or above the specified priority to the fusion engine. All sensors are set to the same ‘Send Above Priority rule’, and all sensors immediately send alerts to the fusion engine that meet the priority criteria. A flowchart depicting the Send Above Priority Rule can be seen in Figure 5-7.

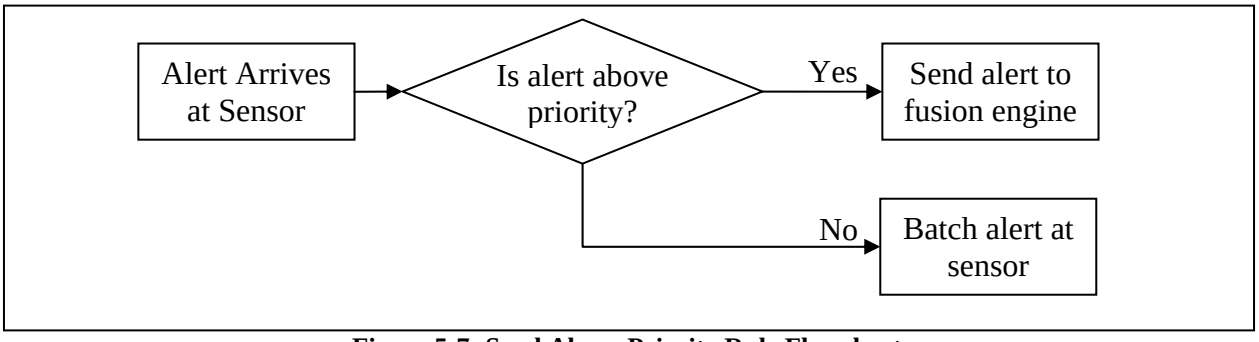

Figure 5-7: Send Above Priority Rule Flowchart

5.2.3 Connectivity Revision

This rule combines the attack tracking capabilities of the Connectivity and Priority rule with the frugalness of the Send Above Priority Rule. In this rule, the sensors are set to external monitor, internal monitor or standby rules just as in the Connectivity and Priority rule, but the external and internal monitor rule are modified slightly. In the Connectivity Revision rule there is no maximum batch size that triggers a sensor in external/internal monitor mode to send all its alerts to the fusion engine. Instead, alerts are only sent to the fusion engine if they are above the current minimum priority of the system, or if the alert is polled by an under capacity or over capacity rule. Due to the relatively low levels of high priority alerts, the Connectivity Revision rule almost always is in an under capacity state. Unlike the other main rules, when the system is under capacity and the Connectivity Revision rule has been chosen, the under capacity rule gives priority to the sensors that are part of the fusion engine wants. Flowcharts for the external and internal monitor rule can be seen in Figure 5-8, while a flow chart for a sensor's actions after the switch from standby to internal monitor mode can be seen in Figure 5-9.

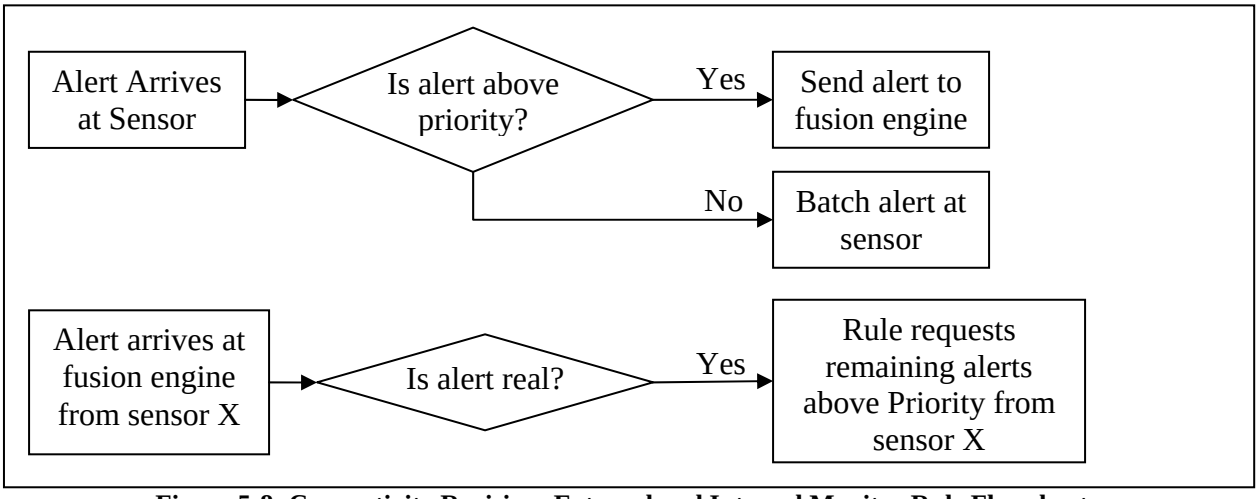

Figure 5-8: Connectivity Revision: External and Internal Monitor Rule Flowcharts

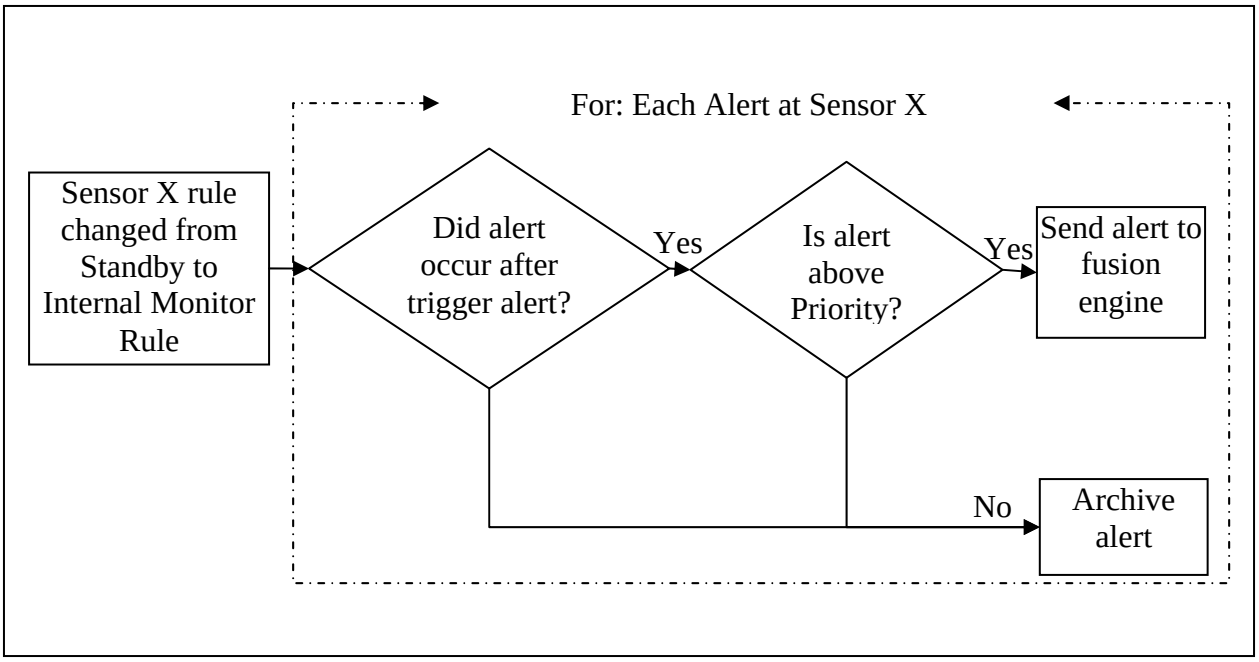

Figure 5-9: Connectivity Revision: Switching from Standby to Internal Monitor Mode Flowchart

5.2.4 Send Everything Rule

Another rule available to determine the fusion engine wants is the Send Everything Rule. Much like the name sounds, this rule tells all sensors to send all alerts immediately to the fusion engine. This rule was included to allow the comparison of the effectiveness of the Connectivity and Priority rule to other more simplistic, although often employed, sensor management strategies. The flowchart of the actions a sensor takes while following the Send Everything Rule can be seen in Figure 5-10.

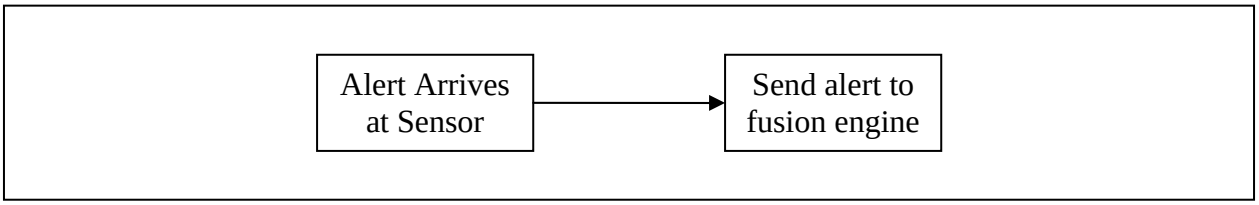

Figure 5-10: Send Everything Rule Flowchart

5.2.5 Send Everything In Batches Rule

Another rule available to determine the fusion engine wants is the Send Everything in Batches Rule. This rule forces the sensors to batch alerts as they are received until the sensor collects a number of alerts equal to the specified minimum batch size. Once a full batch is collected by a sensor, all alerts are sent to the fusion engine. A flowchart depicting the Send Everything In Batches Rule can be seen in Figure 5-11.

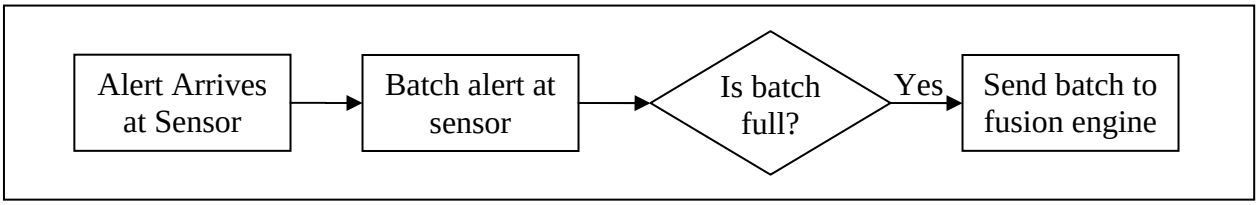

Figure 5-11: Send Everything in Batches Rule Flowchart

5.3 Over Capacity State

An intrusion detection system may be limited by a fusion engine's ability to process alerts as well as the bandwidth available to transmit alerts through the network. If the fusion engine becomes overloaded, a long queue may form leading to unacceptably long delays in the processing of alerts. For a system to remain responsive, the queue at the fusion engine and queue for available bandwidth must remain short, so that each alert can be processed in a timely manner. Rules were developed to attempt to send only the most promising alerts to the fusion engine during periods of over capacity in an attempt to keep the queues from growing uncontrollably long.

Part of the rule specifications is to specify a maximum queue length in units of time for the fusion engine and the available bandwidth. When the fusion engine or bandwidth queue reaches this maximum length, the system enters the over capacity state. When the system is in the over capacity state the system may follow one of several rules that chooses a subset of the

fusion ‘wants’ to send to the fusion engine. Once the offending queue has returned to a predefined minimum queue threshold, the system will again follow the main rule.

Several options are available for choosing the subset of alerts to send to the fusion engine including: priority sampling, edge sampling, random sampling, or increase priority threshold. Alternatively, the user can choose to do nothing in the over capacity situation and continue to send all wants to the fusion engine despite over loading the fusion engine or using too much bandwidth. Each of the rules that involve sampling sample alerts at a rate equal to the tightest constraint of the system. For example, if the system is limited by the fusion engine processing rate, then the constraint rate equals the processing rate of the fusion engine. If the system is limited by the bandwidth, then the constraint rate equals the minimum bandwidth available in the system. Each of the over capacity rules is discussed in detail below.

5.3.1 Priority Sampling Rule

Every ten times the priority sampling rule is called, an initial sampling of all sensors that are part of the fusion engine wants is performed. As an alert is sampled from each sensor, the priority of the alert is recorded by the rule. After the rule samples one alert from each sensor, the sensor with the highest priority alert is selected to send the next alert in the sensor’s queue. The rule continues to select the sensor with the highest priority alert until the constraint rate has been reached for that second, or until no more alerts are available for sampling. In the case of a priority tie between sensors, the rule chooses between the tied sensors randomly. A flowchart depicting the Priority Sampling Rule process is illustrated in Figure 5-12.

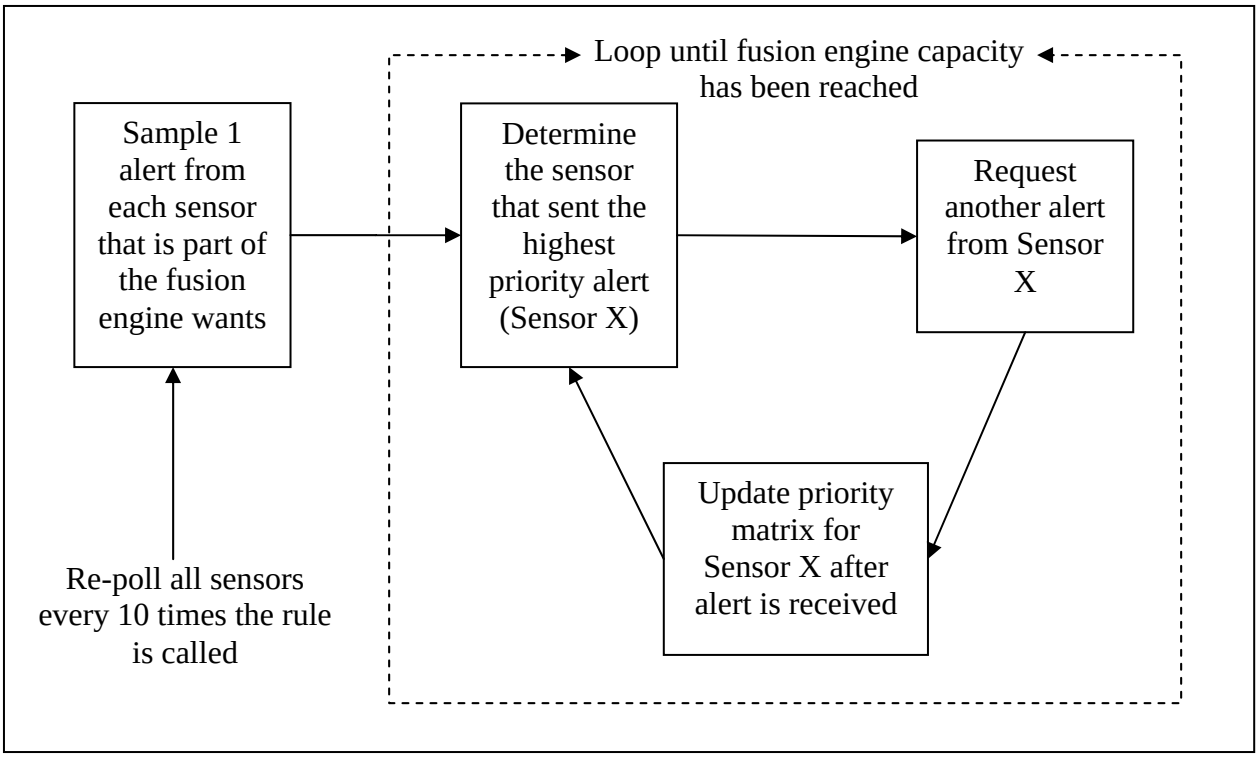

Figure 5-12: Over Capacity Priority Sampling Rule Flowchart

5.3.2 Random and Cyclic Sampling Rules

The random and cyclic sampling rules differ only in how a sensor is chosen by the rule. The random sampling procedure selects a sensor randomly from the sensors that are part of the fusion engine wants, while the cyclic sampling rule cycles through each of the sensors one at a time that are part of the fusion engine wants.

For both the random and cyclic sampling rules the user must specify a queuing priority by which to retrieve alerts from the chosen sensor. Alerts can be retrieved from the sensors by highest priority, by first-in-first-out(FIFO), or by last-in-first-out(LIFO).

Once a random sensor is chosen, an alert is selected from this sensor based on the user's specified queuing policy. If highest priority were chosen as the queuing rule, the highest priority alert is sent to the fusion engine. If there were several alerts that have the highest priority, the most recent alert is sent to the fusion engine. If FIFO were chosen as the queuing rule, the alert

that has been waiting the longest is sent to the fusion engine. If LIFO were chosen, the most recent alert to arrive at the sensor is sent to the fusion engine.

The random and cyclic sampling rules sample alerts each second at a rate equal to the constraint rate of the system, or until there are no remaining alerts to be sampled. A flowchart of the random and cyclic over capacity sampling rules can be seen in Figure 5-13.

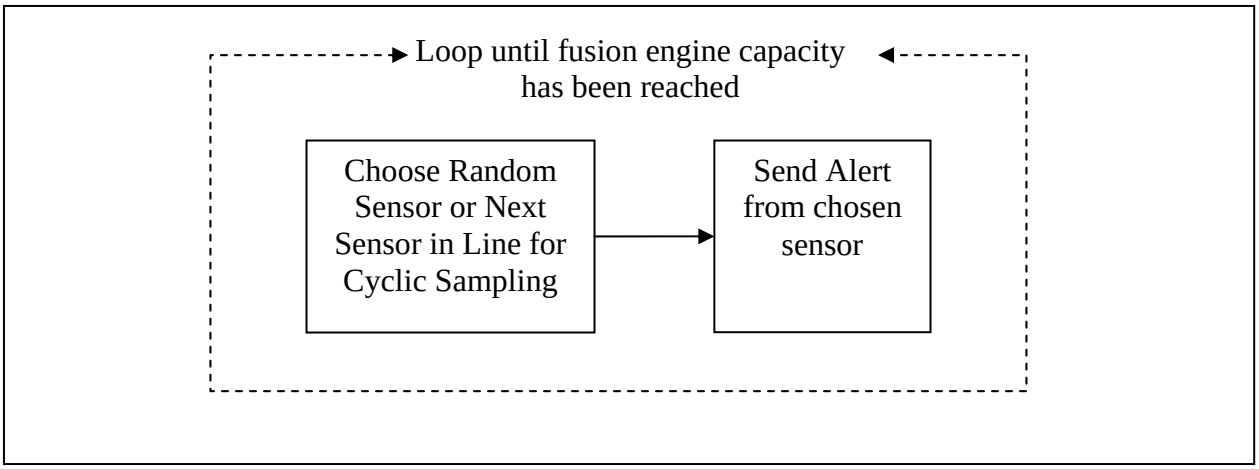

Figure 5-13: Random and Cyclic Sampling Over Capacity Rules

5.3.3 Edge Sampling Rule

The edge sampling rule can only be used in conjunction with the Connectivity and Priority and Connectivity Revision main rules. As attacks progress through a network, more priority is given to the sensors on the perimeter of the attack track and less emphasis is placed on the sensor in the middle of the attack track. The perimeter of the attack track contains all the most recently activated sensors, the sensors that was last added to the attack track, as well as the sensors with external access. In order to quantify the number of alerts sent from each set of sensors, fifteen percent of alerts are sampled from the external access sensors in order to track new attacks. Seventy percent of alerts are sampled from the remaining perimeter sensors, and finally only fifteen percent of alerts are sampled from the sensors in the middle of an attack track.

Similar to the random and cyclic sampling rules, the user must specify the queuing policy at the sensors. An alert may be requested from a sensor using highest priority, FIFO, or LIFO as possible queue choices.

5.3.4 Increase Priority Threshold Rule

The increase priority threshold rule acts differently from the other available over capacity rules. While the other rules focus on sampling alerts from the fusion wants in order to limit the alerts sent to the fusion engine, the increase priority threshold rule adjusts the priority level of the system in order to try to reduce the alerts sent to the fusion engine.

The increase priority threshold rule dynamically adjusts the minimum priority level required for an alert to be sent. If this rule were chosen, the system goes into over-capacity mode when the queue at the fusion or bandwidth reaches the minimum threshold level. When the queue reaches the minimum threshold level the priority level is increased by one point. If the queue were to continue to increase in size, the priority level is also increased. The trigger points for increasing the priority level are two queue lengths evenly spaced between the maximum queue length and the minimum threshold queue length. The priority level is decreased as the queue reduces in size. Decreasing the priority level should allow more alerts to be sent to the fusion engine. The diagram in Figure 5-14 illustrates the position of each of the threshold levels.

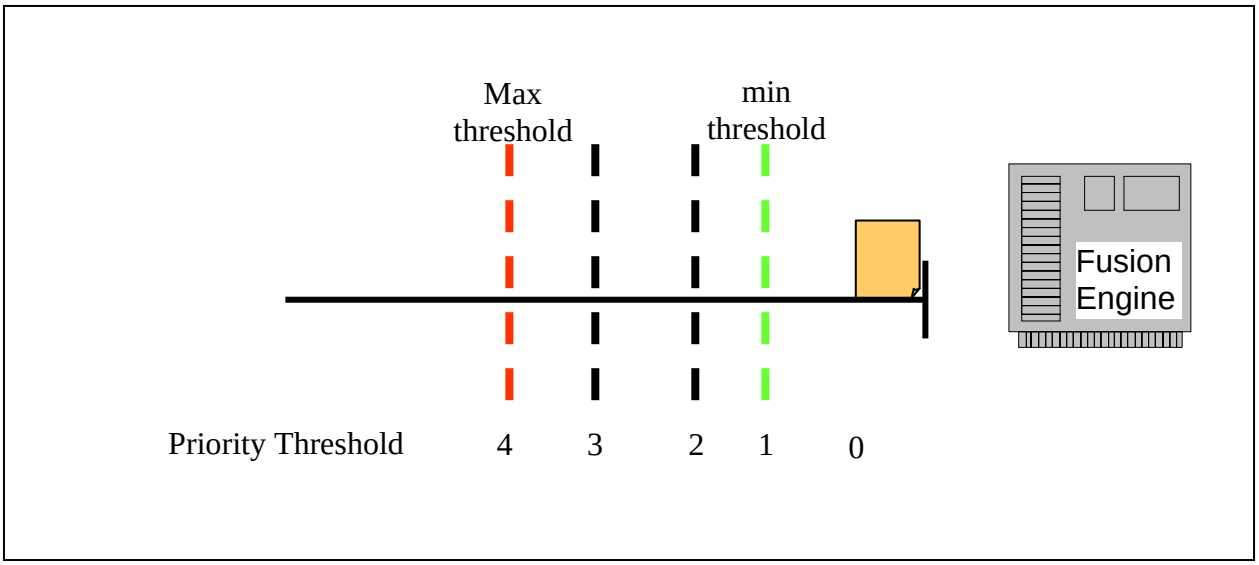

Figure 5-14: Increase Priority Threshold Rule Queue Threshold Diagram

5.3.5 Do Nothing Rule

The do nothing option for the over capacity situation effectively ignores that the bandwidth or fusion engine is being inundated with alerts. Alerts are sent to the fusion engine as they would be if the network were not in an overcapacity state. This may cause the queue at the fusion engine to become very long, and likewise may cause the bandwidth queues at the sensors to become long.

5.4 Under Capacity State

The under fusion capacity state occurs when all fusion engine wants have been sent to the engine, but there is still additional processing capacity remaining at the fusion engine. If this occurs, more alerts can be sent to the fusion engine without impacting the queuing time at the engine. The remaining fusion capacity is filled by sampling from the alerts at all the network sensors. These remaining alerts may reside on any sensor in the network, not only sensors identified as fusion wants. The sampling rate from these sensors is equal to the remaining fusion engine capacity for that second. For example, if the fusion engine can process 30 alerts per

second, and only 20 alerts were sent to the fusion engine in the past second, the rule pulls 10 more alerts to the fusion engine. There are three main rules available for the under capacity state: priority sampling, random or cyclic sampling, and do nothing. Each rule is discussed in the following sections.

5.4.1 Priority Sampling Rule

This rule acts very similarly to the priority sampling rule of the over capacity state. The first time the system requests alerts to fill excess fusion engine capacity, the priority sampling rule does an initial sampling of all the sensors in the network. As an alert is sampled from each sensor, the priority of the alert is recorded by the rule. This initial sampling acts as a guide to help determine which sensor to choose next. The rule requests the next alert from the sensor that delivered the highest priority alert. As each alert is sent to the fusion engine, the rule records the alert's priority. The rule continues to sample the sensors with the highest priority alerts until the excess capacity at the fusion engine is filled. If there is a tie between two or more sensors, a sensor is chosen randomly from among the tied sensors to send an alert.

As the simulation progresses there is a need, at some point, to re-sample from all the sensors in order to maintain a current list of priorities available at each sensor. In order to maintain a more accurate priority list, all the sensors are sampled once every ten times the priority rule is called.

5.4.2 Random and Cyclic Sampling Rules

The random and cyclic sampling rules act very similarly to the random and cyclic rules of the over capacity state. The only differences between the rules in the two states is the sampling base, and the rate of sampling. In the under capacity state the alerts are sampled from all the

sensors in the network, and the sampling rate is equal to the remaining fusion capacity. See section 5.3.2 for a full description of the random and cyclic sampling rules.

5.4.3 Do Nothing Rule

The do nothing rule of the under capacity state effectively ignores that the system is underutilizing the fusion engine. The alerts corresponding to fusion engine wants continue to be sent to the fusion engine, but no additional alerts are sent to the engine to fill the excess available capacity.

6 SIMULATION METHODOLOGY

In order to effectively test the sensor management rules, a simulation environment must model both the cyber domain including network topology and cyber attacks, as well as the sensor management rules. A Cyber Attack Simulator has been created that effectively models cyber networks and attacks on the networks. The simulator allows a user to specify the topology and components of a computer network including the locations of sensors in the network. Once a network is designed, the simulator can be used to create cyber attacks on the network. These simulated cyber attacks trigger sensors in the network to produce alerts corresponding to attack activities. The purpose of the Cyber Attack Simulator is to generate sensor alert datasets that are then used to test the effectiveness of information fusion engines at tracking cyber attacks through computer networks (Kistner, 2006, Costantini, 2007, Kuhl et. al. 2007).

The Cyber Attack Simulator is an object oriented discrete event simulation implemented in Java. Since the sensors were modeled as individual objects in the Cyber Attack Simulator an additional package was built to control the sensors during simulation runtime. The package adds sensor management capabilities to the Cyber Attack Simulator so that the sensors could be controlled during the simulation of the cyber attacks. In order to accurately model the sensor management capabilities several new classes are created to allow the modeling of an information fusion engine, bandwidth constraints, and the sensor management rules.

6.1 Sensor Management Package

To test the sensor management rules a sensor management package was created to interface with the Cyber Attack Simulator. The sensor management package controls the sensors during a scenario run. This new package includes extensions of existing classes such as the

sensor and traffic classes, as well as several new classes in order to model the fusion engine, the rule itself, and to facilitate control of the sensors. This section covers the sensor management package architecture including how the classes interact with one another, as well as how the sensor management package interacts with the Cyber Attack Simulator. Additionally, the details of the main sensor management package classes are explained. Names of Java classes will appear in italics.

6.1.1 Architecture Overview

There are two main ways that the sensor management package interacts with the Cyber Attack Simulator. The primary interaction occurs when the rule sensors receive alerts as they are generated by the simulator. This interaction is seen in Figure 6-1 as the arrow for alerts pointing from the *attack scenario* class to the *rule sensor* class. The receipt of an alert triggers the sensor to follow the rule the sensor has been assigned by the *rule* class to determine what to do with the alert the sensor has received. The second interaction occurs through the simulation event table, or the *event order* class. This second interaction can be seen in Figure 6-1 as the arrows that point from the *fusion engine traffic* and *bandwidth traffic* classes to the attack scenario's *event order* class. The sensor management package adds events to the scenario event table in order to control the processing of alerts at the fusion engine, and to determine bandwidth and resource usage in the network. In order to add events to the event table two extensions of the traffic class were created, *bandwidth traffic* and *fusion engine traffic*. Adding events to the event table allows the sensor management package to operate with the same simulation clock that is being used to generate the alerts.

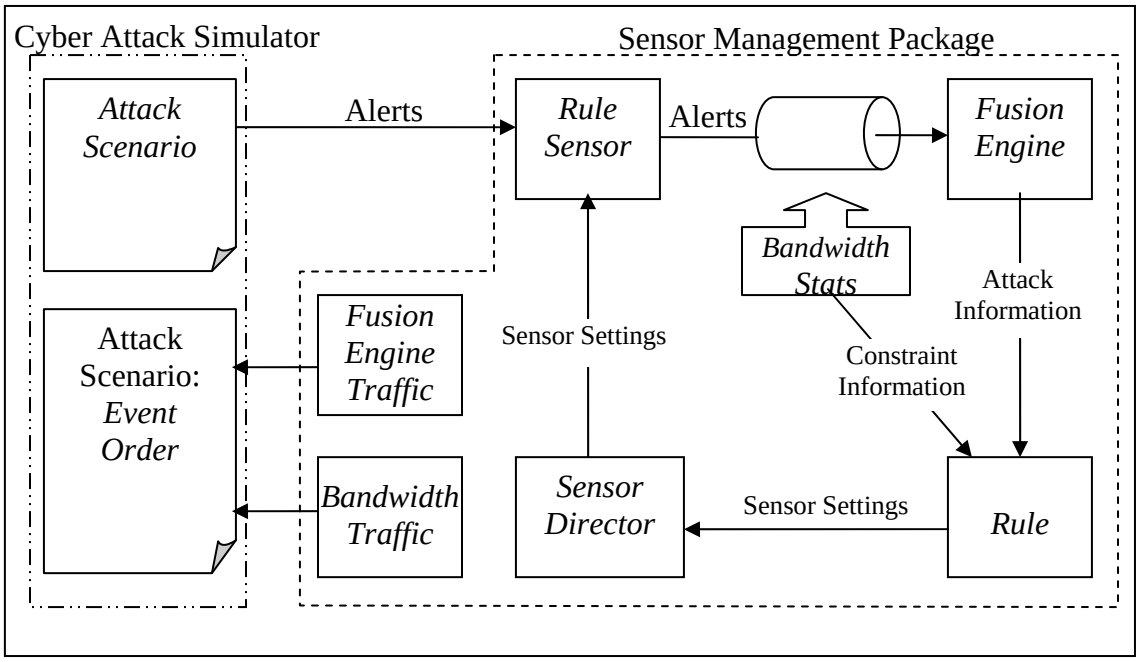

Figure 6-1: Sensor Management Architecture

Here a brief overview of the sensor management architecture is given prior to developing the details of each class. As shown in Figure 6-1, the main classes of the sensor management package include the *rule sensor* class, *fusion engine* class, *bandwidth stats* class, *rule* class, and *sensor director* class. These classes all work together in order to control the action a sensor takes once an alert is received. If the sensor decides to send an alert to the fusion engine, the sensor must first check with the *bandwidth* class to determine if sending the alert will not exceed any constraints in the network. Once the *fusion engine* receives alerts, they are processed to determine if the alert corresponds to any active attacks on the system. If the *fusion engine* correlates an alert to an attack track, the updated attack information is sent to the *rule* class. The *rule* class uses information from the *fusion engine* as well as constraint information from the *bandwidth* class to decide what rules each individual sensor should be following. The *rule* class sends the sensor setting assignments to the *sensor director* whose job is to change each sensor's rule setting to the newly chosen rules.

6.2 Sensor Management Package Class Descriptions

The sensor management package add-on to the Cyber Attack Simulator consists of several extensions to existing classes as well as several new classes, and the package provides the primary simulation framework to test the sensor management rules. The important details of each of the sensor management package classes are discussed in the following sections.

6.2.1 Rule Sensor Class

The *rule sensor* class is an extension of the *sensor* class. An instance of the *rule sensor* class has all the same properties and functionality as the original sensors with additional properties that are used to control the action of the sensor once an alert is received. Once an alert arrives at a rule sensor, the rule sensor will decide what to do with the alert based on the rule the sensor has been assigned by the *rule* class. A rule sensor may send an alert to the fusion engine, store the alert to send later, or ignore the alert altogether.

A rule sensor is triggered to send an alert to the fusion engine by one of two events. The first event is the arrival of an alert to the sensor. When an alert arrives at a rule sensor, the sensor is triggered to process the alert according to the sensor's current rule setting. The current rule may cause alerts to be sent to the fusion engine. The second event is a pull mechanism by which the *rule* class may sample alerts available at a sensor and tell a sensor to send a specific alert to the fusion engine.

An instance of a *rule sensor* keeps track of statistics important to the overall performance of a rule. These statistics include: the number of alerts received, the number of alerts sent to the fusion engine, the number of real alerts received, the number of noise alerts received, the number

of alerts received while the sensor was in standby mode, and the time in queue a real alert waits at a sensor before being sent to the fusion engine.

The rule sensors have the ability to keep track of all the alerts received and all the alerts sent to the fusion engine. At the end of a simulation run, two files can be created for each sensor, one file contains a list of all the alerts generated by that sensor while the second file creates a list of all the alerts sent to the fusion engine. When running scenarios that may generate large numbers of alerts, the user may choose to disable the saving of alerts to allow the simulation to run faster as well as avoid memory problems. The pertinent statistics with regards to rule performance are saved regardless of whether the alerts are saved or not.

6.2.2 Rule Class

The *rule* class is a static class that receives information from the *fusion engine* and the constraints of the network in order to determine what rule each sensor should have and what sampling plan, if any, should be activated in the network. The *rule* class has three main responsibilities. The first responsibility is to set up the sensors by assigning the sensors their starting rule, or default rule, based on the overall rule schema chosen by the user. The second responsibility is to dynamically assign rules to each sensor based on the progression of attacks in the network and the chosen rule schema. The third responsibility is to sample alerts from sensors as the need arises. Detailed information on the actual rules that the *rule* class uses is in the previous section.

The *rule* class stores information on the attacks that are in progress in attack sets. For each attack in progress, the *rule* class maintains a list of the machines involved in the attack, as

well as the settings of all sensors involved in the attack. These attack sets are used to reset sensors in the network after an attack has been completed.

The *rule* class has several parameters that can be adjusted by the user before each run. The parameters include the rules that will be used during the run, the maximum batch size allowed at a sensor, the bit size of an alert, whether or not the alerts for a scenario should be saved by the sensors, and minimum and maximum fusion engine queue size thresholds.

6.2.3 Sensor Director Class

The *sensor director*, like the *rule* class, is a static class. The *sensor director* is primarily responsible for changing the rule that each sensor is following based on the assignments made by the *rule* class. The *sensor director* is the class that interacts with the sensors in order to change their settings. A secondary responsibility of the *sensor director* is to reset all statistic collection fields at the beginning of a new simulation run.

6.2.4 Fusion Engine and Fusion Engine Traffic Classes

The *fusion engine* class represents an information fusion engine that works with 100% accuracy. An instance of the *fusion engine* class maintains a queue of alerts that need to be processed. As a fusion engine processes an alert, the fusion engine first determines if the alert is real or noise. A real alert is an alert that was created as a result of an attack in the attack scenario. A noise alert is a randomly generated alert that was not caused by an attack in the attack scenario. If the alert is real, the alert must then be matched to an existing attack track. If the alert does not belong to an existing attack track, then a new attack track is created. The real alert, as

well as the target machine of the alert, and the time of the original alert are sent to the *rule* class to be processed.

One important attribute of the *fusion engine* class is the processing rate. The processing rate of the fusion engine is specified in alerts per second and can be controlled by the user at runtime. The processing rate of the fusion engine determines how fast alerts are processed in the queue. If the queue at the fusion engine becomes too long then the *rule* class is notified. The rules followed by the sensors as well as any sampling procedures may be modified until the queue at the fusion engine goes down to a more desirable level.

As the fusion engine processes alerts additional *fusion engine traffic* objects are added to the attack scenario event table. This allows the fusion engine to process the alerts as the simulation is running. After an alert has been processed, the fusion engine will add the next traffic step to the event table if there is an alert waiting in the fusion queue.

6.2.5 Bandwidth Traffic and Bandwidth Stats Classes

The *bandwidth traffic* class is used to put a reoccurring event in the event table. This event recalculates the average and max bandwidth used per network link each second of the simulation. The *bandwidth stats* class is a static class in charge of actually performing the bandwidth calculations. Each time the *bandwidth stats* class recalculates the bandwidth performance measures, the class also checks the state of the system to determine if the system is over capacity or under capacity.

6.3 Using the Sensor Management Package and Cyber Attack Simulator

There are several new interface components that a user encounters when using the Cyber Attack Simulator with the sensor management package. The new interfaces allow a user to turn on the sensor management functionality, place a fusion engine in the network, edit the fusion engine's properties, and set up sensor rules. The new interface features are discussed below.

6.3.1 Starting the Sensor Management Package

The Java Cyber Attack Simulator does not automatically use the Sensor Management Package. The Sensor Management Package must be turned on when a network is created. To turn on the Sensor Management Package click the checkbox next to "Sensor Management Functionality" in the New Network form as pictured in Figure 6-2.

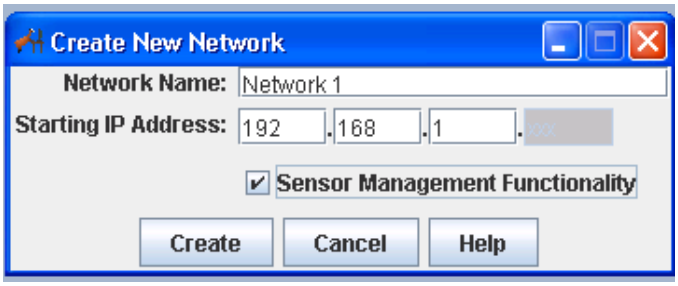

Figure 6-2:Sensor Management Checkbox on New Network Form

Turning on the Sensor Management Package adds a new menu called "Fusion" to the menu bar, and a new button called "Place Fusion Engine" next to the other network buttons. The Fusion menu and the "Place Fusion Engine" button can be seen in Figure 6-3.

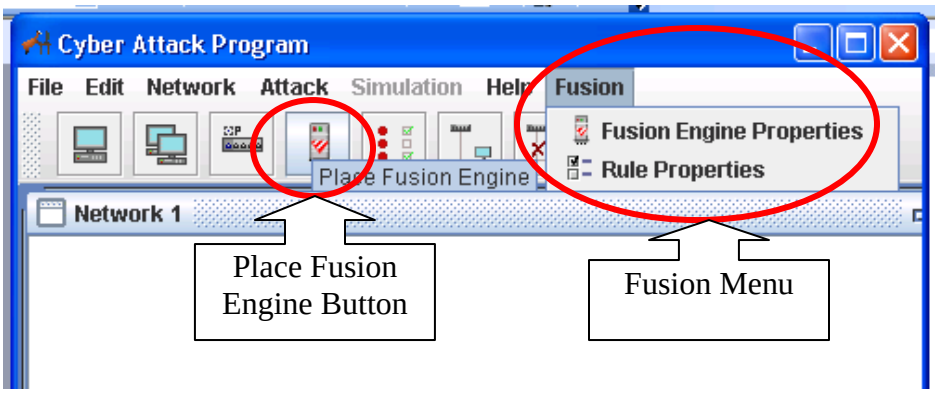

Figure 6-3: Sensor Management Features

6.3.2 Editing the Fusion Engine

When creating a network using the Sensor Management Package, an additional step must be taken to place the fusion engine somewhere in the network. Placing the fusion engine can be done by clicking the "Place Fusion Engine" button, or selecting "Fusion Engine Properties" from the Fusion menu. A form appears as in Figure 6-4.

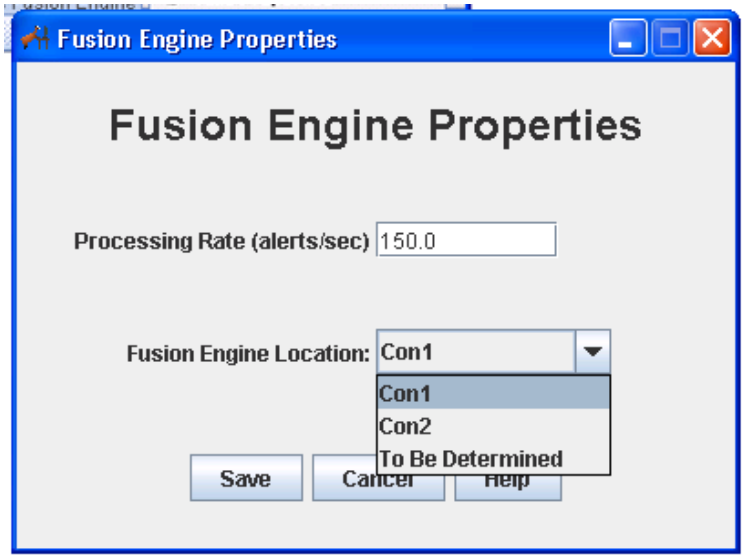

Figure 6-4: Fusion Engine Properties Form

The fusion engine can be placed on any of the connectors in the network by choosing the desired connector in the fusion location drop down menu. In addition to specifying the fusion engine location, this form can be used to update the fusion engine processing rate. Once the

fusion engine has been added to the network, the engine appears in the network frame as pictured in Figure 6-5.

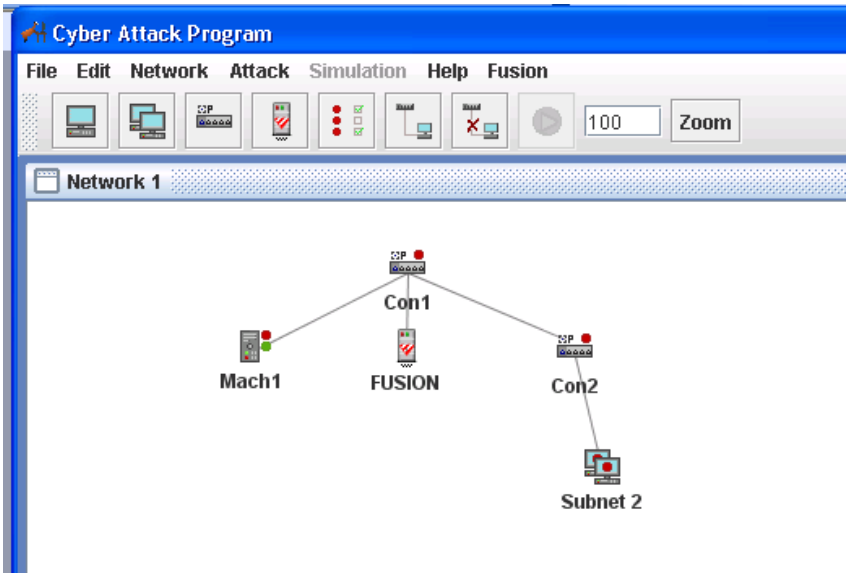

Figure 6-5: Cyber Network With Fusion Engine

6.3.3 Choosing the Sensor Management Rules

The user has the ability to set up the sensor management rules using the "Rule Properties" form. The "Rule Properties" form is accessed through the Fusion menu and is seen in Figure 6-6. This form is used to choose the rule that will be applied in each of the system states as well as to specify supporting information for each rule that is chosen. For the main fusion wants rule, the user must choose a rule, and may additionally need to specify the minimum alert priority that can be sent to the fusion engine as well as the maximum batch size of alerts allowed at a sensor. Depending on the under capacity rule chosen, the user may need to specify the queuing priority. When choosing the over capacity rule the user again may need to specify the queuing priority depending on what rule is chosen, and additionally the user needs to specify the maximum and minimum queue lengths at the fusion engine. These queue lengths are used to evaluate the state of the system.

The Rule Properties form is also used to specify whether the alerts generated by each sensor should be saved. The default setting is that sensor alerts are not saved. Not saving the sensor alerts allows the simulation to run much faster. If the user needs the alerts generated at each sensor, the "Save Alerts" check box can be checked and the alerts are automatically saved by the program at the end of each scenario run.

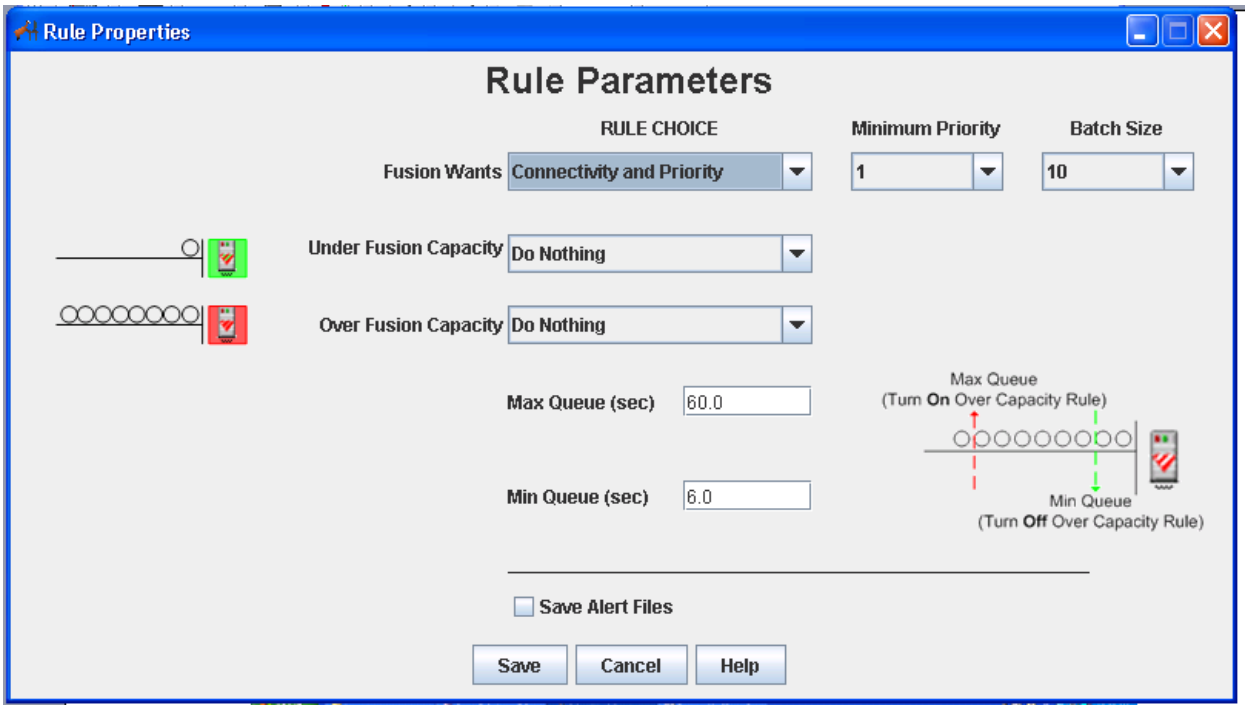

Figure 6-6: Rule Properties Form

6.3.4 Viewing Results

When a scenario is run with the sensor management package in use, a results page appears on the screen at the end of the scenario. This results page provides a summary of the scenario. A text file is created that is saved in the network's results folder. This text file can be opened in Excel and has all the summary information, plus additional information on bandwidth use for each link in the system and the number of alerts processed versus total alerts generated for each attack in the scenario.

7 A STUDY OF EVENT BASED DISPATCHING RULES

This section outlines the design of the experiment used to test the effectiveness of the various sensor management rules developed. First, an overview of the entire experiment will be discussed, followed by a detailed description of the individual experimental factors and the performance measures recorded for analysis.

7.1 Methodology

The purpose of this experiment is two-fold. The first purpose is to identify a robust sensor management strategy that performs well for a variety of performance measures and network configurations. The second purpose is to identify characteristics of a computer network that may affect the performance of sensor management rules. This goal of this experiment is to identify rules that both perform well for a comprehensive set of performance measures and perform robustly across a variety of network configurations.

7.1.1 Experimental Design

In order for a sensor management strategy to be considered useful, the strategy should work consistently and effectively for a variety of network configurations and attack patterns. Networks can differ in many ways including the overall layout of the network or network topology, the number of machines or size of the network, and the overall volume of traffic and noise generated within the network. In this experiment the network topology, the number of computers, and the alert rate of the network are chosen as the experimental factors along with all the practical rule combinations.

Network topology may be a significant factor in the performance of the connectivity based rules. As the number of layers in the network increases, the performance of the connectivity based rules could improve over global network rules such as the Send Everything Rule. As the number of layers in a network increases, the connectivity rules have more opportunities to manipulate the sensor settings as an attack is tracked through the network. Multiple network layers provide natural protective layers to sensors deeper within the network. The sensors on the unthreatened network machines can be kept in standby mode, thereby preventing unwanted noise sent to the fusion engine. When there are fewer layers, more computers are vulnerable to attack, so the connectivity rules could have little advantage over the global rules. Three different network topologies are tested, each with increasing numbers of layers within the networks: a one layer network, three layer network, and five layer network. These networks are pictured and discussed in the Test Networks section below.

The number of computers in the network may affect the performance of some of the rules, particularly the rules involving cyclic and random sampling. As the number of machines in the network increases, so may the number of sensors. With more sensors in the network, when the sensors are polled in a cyclic manner, the time taken to return to each sensor may increase, and this increase in polling interval may increase the time real alerts spend waiting at a sensor before they are sent to the fusion engine for processing. Likewise, an increase in the number of sensors decreases the chance that a certain sensor is polled during the random polling of sensors. A sensor that is not polled for an extensive period of time may lead to a high wait time for alerts at these sensors. For each of the three network topologies created, there is a small network and a large network version. The small network contains 250 machines distributed evenly throughout

the subnets, while the large version has 1000 machines distributed evenly throughout the network subnets.

The third test factor that may affect the performance of the sensor management rules is the volume of alerts in the system with respect to the capacity of the fusion engine. If the rate of alerts in the network is significantly less than the capacity of the fusion engine then a sensor management strategy is not even necessary because the fusion engine can easily process all alerts in the system in a timely manner. Sensor management strategies become important as the rate of alerts in the network approaches or exceeds the processing rate of alerts at the fusion engine. The rules that effectively limit the number of alerts sent to the fusion engine to less than or equal to the fusion engine processing rate will work well when the rate of alerts arriving exceeds the capacity of the fusion engine. Rules that ignore the capacity of the fusion engine may develop large queues at the engine, and the fusion engine may not be able to process real alerts in a timely manner. Two different alert levels are used in this experiment, a low level and a high level. The low level corresponds to roughly 100% of the fusion engine capacity while the high level corresponds to 115% of the fusion engine capacity.

The final factor in this experiment is all the practical combinations of the developed rules. For each simulation run, a Main Rule, Under Capacity Rule and Over Capacity Rule were chosen. Table 7-1 summarizes the available combinations for each of the Main Rules. There are a total of 58 rule combinations. An initial set of all 58 rule combinations is run. From the initial set, a subset of the rules is chosen for analysis. The subset of rules contains all rules whose performance is significantly different from another rule.

There are several parameters that these rules require. For every scenario the maximum fusion engine queue is set to 60 seconds, while the threshold queue length is set to 6 seconds.

The fusion engine capacity is set to 100 alerts per second in each scenario. For every rule that requires batching, a batch size of 10 alerts is used. For every rule that requires a queue choice of FIFO, LIFO or highest priority, the highest priority rule is chosen. For rules that require the specification of a minimum priority, a priority level of 1 is chosen.

Table 7-1: Experiment Rule Combinations

Main Rule	Under Capacity Rules	Over Capacity Rules	Combinations
Connectivity and Priority (CP)	Do Nothing (DN) Random Sampling (RS) Cyclic Sampling (CS) Priority Sampling (PS)	Do Nothing (DN) Edge Sampling (ES) Random Sampling (RS) Priority Sampling (PS) Increase Priority Threshold (IPT)	20
Connectivity Revision (CR)	Random Sampling Cyclic Sampling Priority Sampling	Do Nothing Edge Sampling Random Sampling Priority Sampling Increase Priority Threshold	15
Send Everything in Batches (SEB)	Do Nothing Random Sampling Cyclic Sampling Priority Sampling	Do Nothing Random Sampling Priority Sampling Increase Priority Threshold	16
Send Everything (SE)	Do Nothing	Do Nothing Random Sampling Priority Sampling Increase Priority Threshold	4
Send Above Priority (SAP)	Random Sampling Cyclic Sampling Priority Sampling	Do Nothing	3
		TOTAL	58

A summary of the experimental factors and levels can be seen in Table 7-2. Four replications of a test scenario are completed for each set of conditions. There are a total of 2784 experimental runs.

Table 7-2: Experiment Factor Summary

Factors	Levels
Network Topology (Layers)	1, 3, 5
Network Size (Machines)	250, 1000
Alert Rate (% of Fusion Capacity)	100, 115
Rules	58 Combinations (See Table 7-1)

7.1.2 Test Networks

For security reasons, organizations do not typically make their network topologies available to the public. This experiment uses three network topologies designed to represent increasingly complicated network structures that one may encounter in industry. Although not actual networks, the networks are representative of various network structures encountered in practice. The test networks are referred to by their number of subnet layers and include a one, three, and five layer network.

The one layer network contains three servers with external access, and five subnets accessible from the three servers. Once a server is compromised all the computers in the one layer network become vulnerable to attack. In the small version of the single layer network each subnet contains roughly 49 computers. In the large version of the single layer network each subnet contains 199 computers. The fusion engine is arbitrarily located on connector 6. The layout of the one layer network can be seen in Figure 7-1.

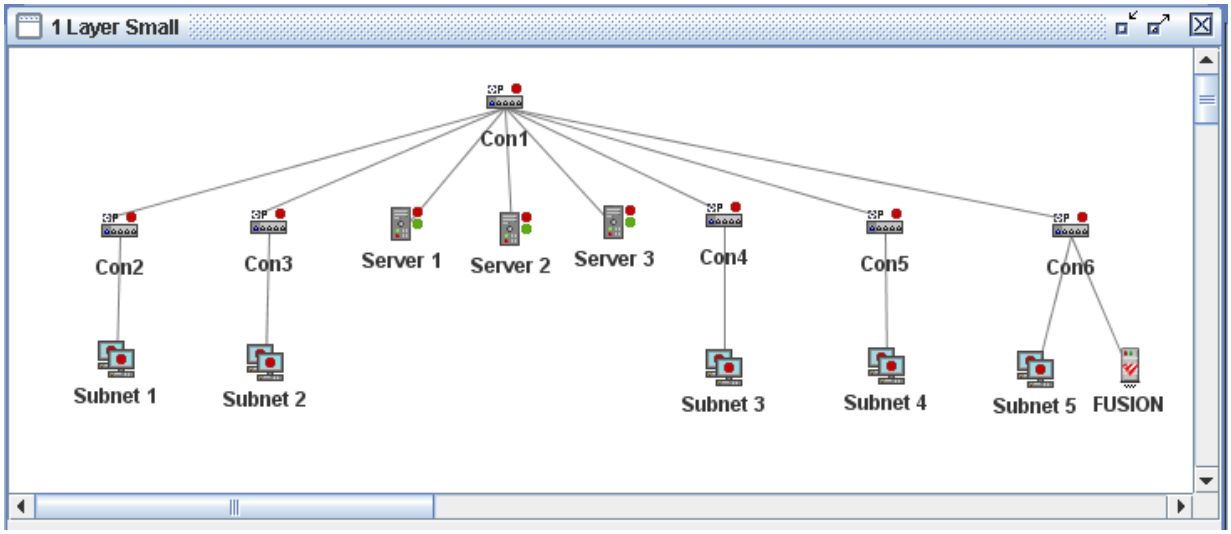

Figure 7-1: One Layer Network Topology

The three layer network provides a test framework where target machines can be several layers deep in the network. The three layer network contains two main branches accessible from three servers with external access. Each branch of the network contains three layers of subnets, for a total of six subnets. The small version of the three layer network contains roughly 41 computers in each subnet, while the large version contained 84 computers per subnet. The fusion engine resides deep within the network on connector 7. The three layer network can be seen in Figure 7-2.

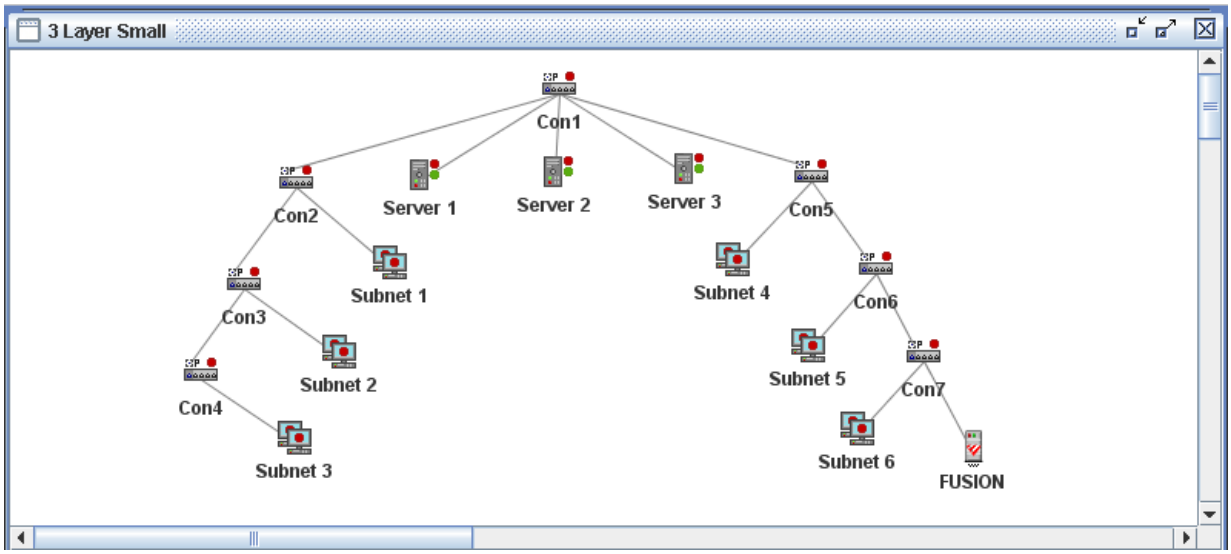

Figure 7-2: Three Layer Network Topology

The branching and multiple layers of the five layer network was designed to represent a complicated network structure that could be found in a large government or industry organization. The five layer network contains many branches and is best introduced by referring the reader to Figure 7-3. The five layer network contains four main branches accessible from three external servers. There are a total of 20 subnets. In the small five layer network each subnet contains roughly 12 computers, while in the large version of the network each subnet contains 49 computers. The fusion engine resides deep within the network on connector 21.

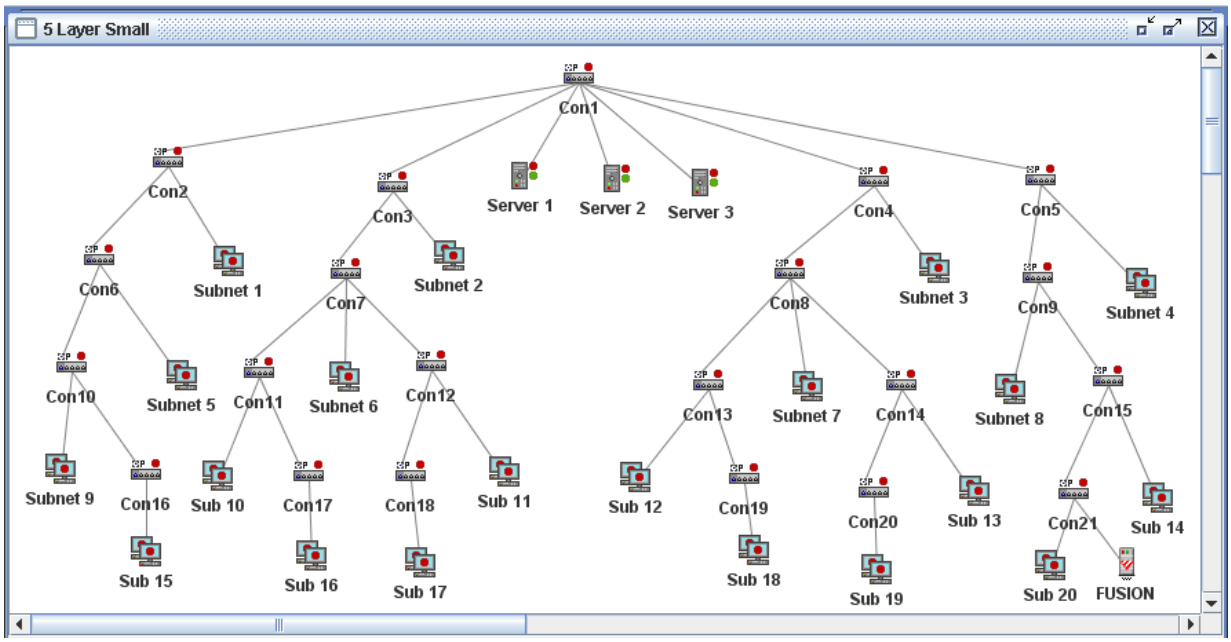

Figure 7-3: Five Layer Network Topology

7.1.3 Test Scenario and Noise Traffic

A test scenario is developed to create a variety of attack conditions that may occur during any given day of network monitoring. The scenario is 12 hours in length and contains ten unique attacks. Once the attacks are complete the scenario runs for an additional 7 minutes. The attacks are dispersed throughout the day so that there are periods of time with no attacks in progress, one attack in progress, or several attacks in progress at the same time. A scenario timeline with the details of each attack can be seen in Figure 7-7. This scenario is used for all three networks with the exception that the target machines for each attack are different for each of the networks. The target subnet for each of the attacks in each of the different networks can be seen in Figures 7-4 through 7-6.

In addition to the attacks in the scenario, the scenario also has a set of noise parameters. The noise contains 98% reconnaissance alerts, 1% intrusion alerts, and 1% escalation alerts. The noise rate of the scenario is manipulated in order to reach the desired overall network alert rates.

For an alert rate of 100% of the fusion engine capacity, the noise rate is set at 175,000 alerts per hour. This corresponds to a noise alert to real alert ratio of roughly 45:1. For an alert rate of 115% of the fusion engine capacity, the noise rate is set to 200,000 alerts per hour. The higher rate corresponds to a noise alert to real alert ratio of roughly 55:1. The fusion engine capacity is set to 100 alerts per second in all network configurations.

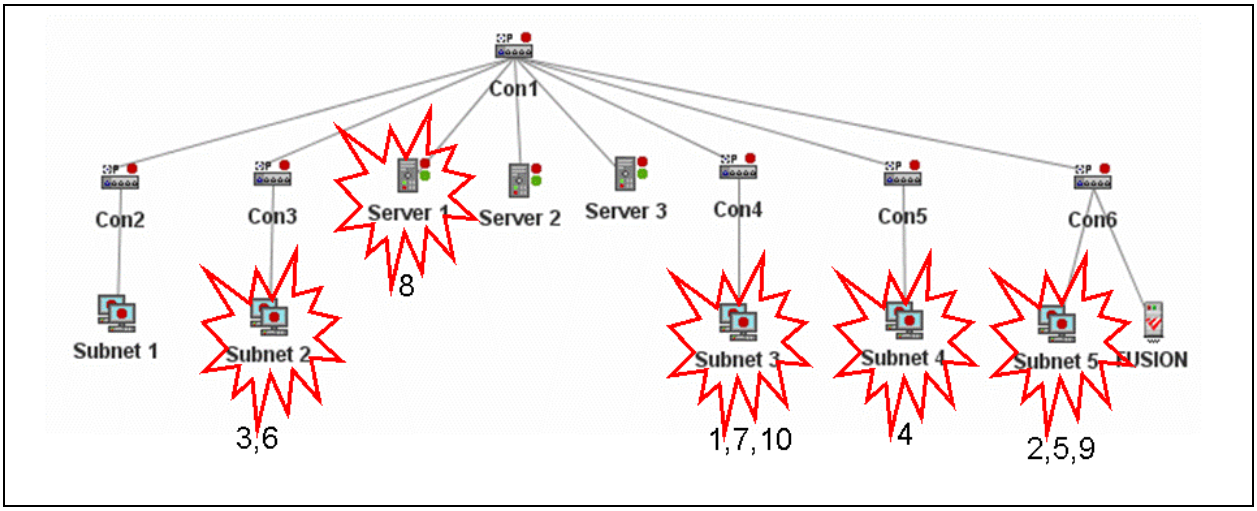

Figure 7-4: One Layer Network Attack Locations

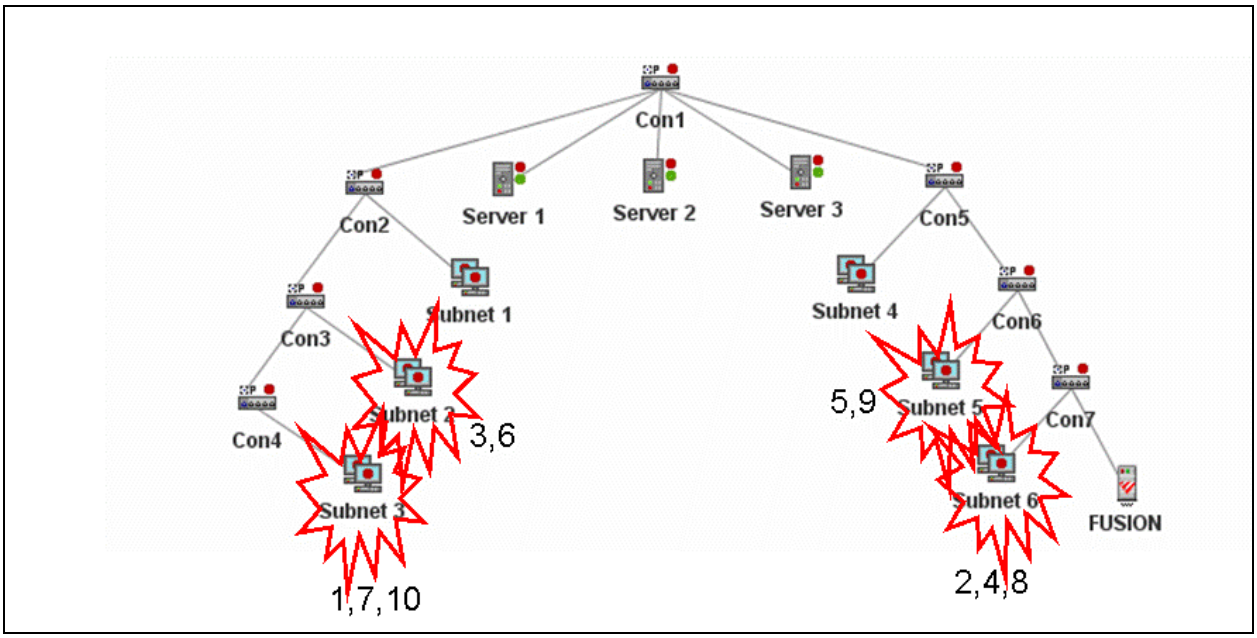

Figure 7-5: Three Layer Network Attack Locations

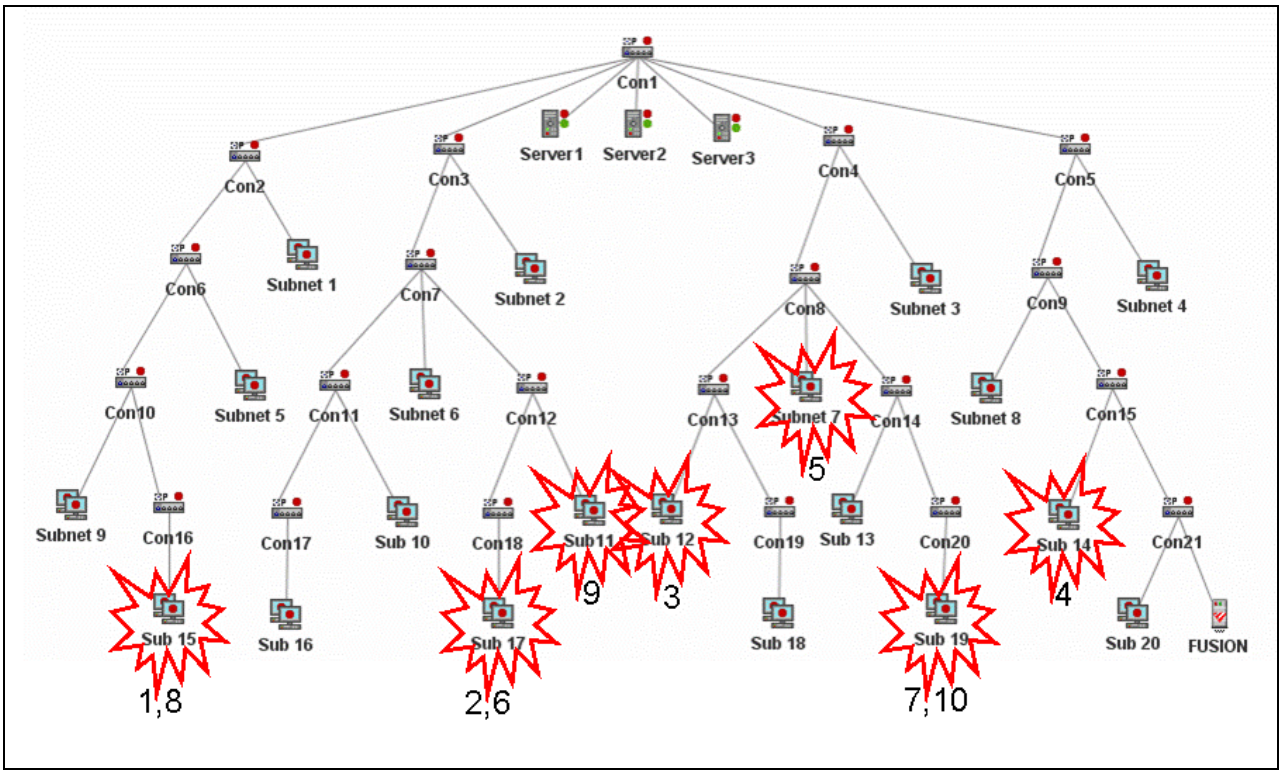

Figure 7-6: Five Layer Network Attack Locations

The Cyber Attack Simulator uses unique random number streams to generate the attack steps for each attack and the noise in each scenario. In order to make four replications of each network configuration and rule combination, four different seeds are chosen for the random number generator. Each seed produces a different set of attacks with the parameters specified in Figure 7-7 and a different set of noise alerts. The random seeds chosen are 456, 57913, 135790, and 8755241.

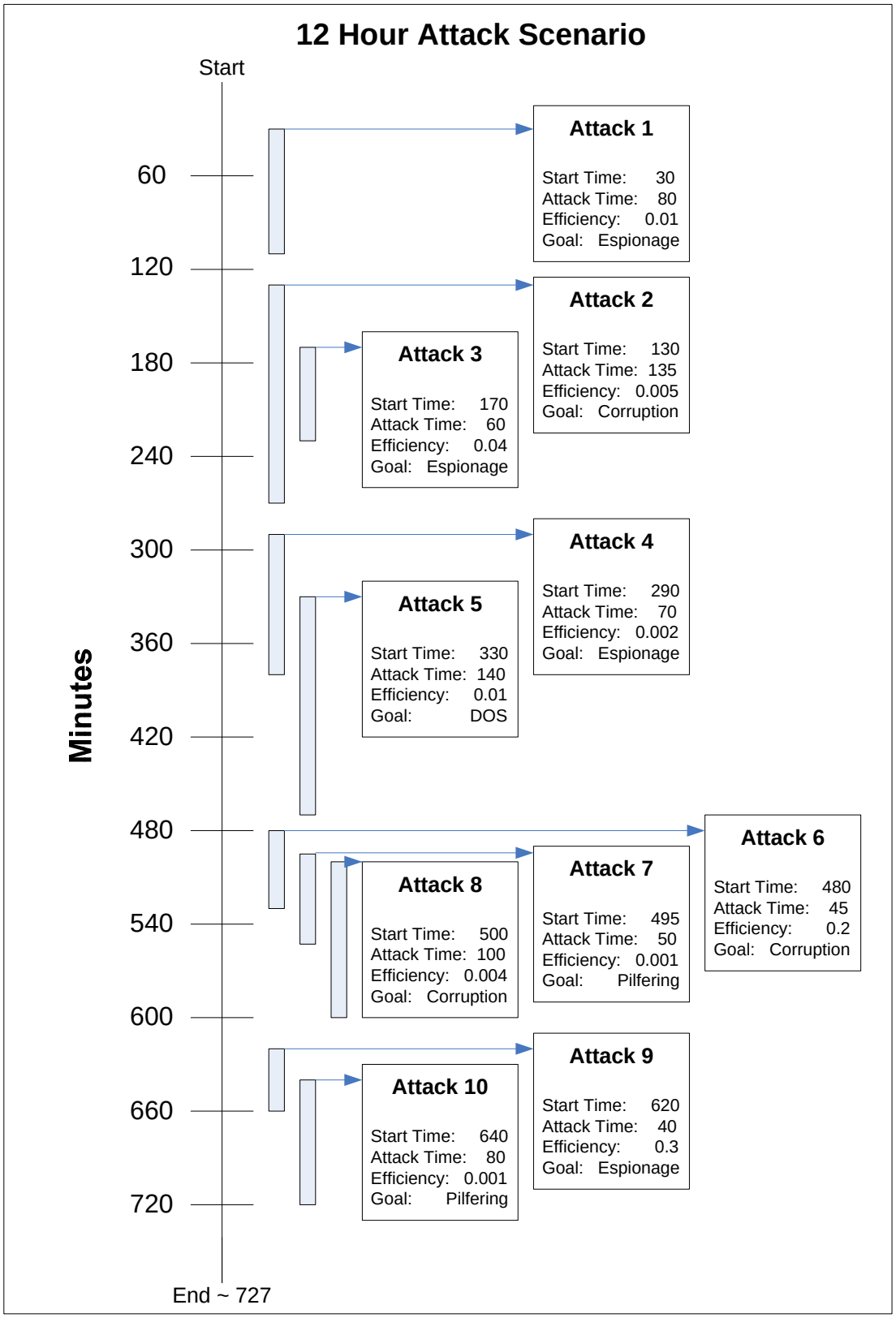

Figure 7-7: Attack Scenario Time Line

7.1.4 Performance Measures

Evaluating the performance of a set of rules requires looking at the rule performance from a variety of perspectives. Looking at several different performance measures is important because some rules may perform well for a subset of performance measures, but very poorly for others. Analyzing multiple performance measures highlights rules that perform well for many if not all of the measures.

The performance measures that are recorded for each scenario replication include:

1. Percentage of all real alerts processed at the fusion engine by the end of the scenario;
2. Percentage of all noise alerts processed at the fusion engine;
3. Average percentage of alerts per attack processed at the fusion engine;
4. Standard deviation of percentage of alerts per attack processed at the fusion engine;
5. Average total time a real alert spends in the system; and
6. Average bandwidth used on the fusion engine link.

A good rule should send a high percentage of the real alerts to the fusion engine for processing. The percentage of all real alerts processed at the fusion engine performance measure applies only to those alerts that are processed by the end of the scenario. For example, a rule may send nearly all of the real alerts to the fusion engine, but due to a queue build up at the fusion engine, not all of the real alerts may be processed by the end of the scenario.

The percentage of all noise alerts processed at the fusion engine measures the amount of noise processed at the engine in relation to the total noise in the system. Processing less noise alerts in exchange for processing more real alerts could indicate that a rule is performing well.

During the scenario run, the number of alerts generated for each attack is tracked along with the number of alerts sent to the fusion engine for each attack. These two measurements allow for the calculation of the percentage of alerts processed for each attack. The average percentage of alerts processed per attack gives an indication of the thoroughness of each rule set at tracking the attacks. The standard deviation of the percentage of alerts processed per attack gives an indication of the ability of a rule set to track all attacks evenly. A rule performing well should have a high average percentage of alerts processed per attack and a low standard deviation of the percentage of alerts processed per attack.

The time a real alert spends in the system is a critical performance measure of the rules. The time a real alert spends in the system should be small. The fusion engine should be able to track attacks in the network in near real time, so that the system administrator or threat assessment tools have an adequate amount to time to react to the attacks on the network while the attacks are still in progress.

Finally, the amount of bandwidth used by a sensor management rule should not be excessive. The amount of bandwidth used by two similarly performing rules may serve as a way to choose between the two rules.

7.2 Results

A summary of the results of the designed experiment are presented in this section. Once all the results were gathered an observation was made that either the under capacity or the over capacity rule played a dominant role in the performance of each of the main rules. For the Connectivity Revision rule and the Send Above Priority rule the under capacity rule was the dominant factor in the rule's performance. For example, for the Connectivity and Priority rule,

all rule combinations that contained cyclic sampling as the under capacity rule performed similarly. For the Send Everything In Batches and Connectivity and Priority rules the over capacity rule acted as the dominant rule. Due to the dominance of one of the under or over capacity rules, only a subset of the entire rule set tested is shown and discussed in this section. For the complete data set see Appendix II. The rules that are the focus of the rest of the results are summarized in Table 7-3. The rules are listed in order of their performance for the time a real alert spends in the system performance measure; the best performing rule is listed first.

Table 7-3: Rule Set Used in Results Discussion

Rule Abbreviations	Main Rule	Under Capacity Rule	Over Capacity Rule
CR-CS-DN	Connectivity Revision	Cyclic Sampling	Do Nothing
SAP-CS-DN	Send Above Priority	Cyclic Sampling	Do Nothing
SAP-RS-DN	Send Above Priority	Random Sampling	Do Nothing
SAP-PS-DN	Send Above Priority	Priority Sampling	Do Nothing
CR-RS-DN	Connectivity Revision	Random Sampling	Do Nothing
CR-PS-PS	Connectivity Revision	Priority Sampling	Priority Sampling
CR-PS-DN	Connectivity Revision	Priority Sampling	Do Nothing
CP-DN-RS	Connectivity and Priority	Do Nothing	Random Sampling
CP-DN-ES	Connectivity and Priority	Do Nothing	Edge Sampling
SE-DN-RS	Send Everything	Do Nothing	Random Sampling
SEB-DN-RS	Send Everything Batches	Do Nothing	Random Sampling
CP-DN-PS	Connectivity and Priority	Do Nothing	Priority Sampling
CP-DN-DN	Connectivity and Priority	Do Nothing	Do Nothing
SE-DN-IPT	Send Everything	Do Nothing	Increase Priority Threshold
SEB-DN-IPT	Send Everything in Batches	Do Nothing	Increase Priority Threshold
SE-DN-DN	Send Everything	Do Nothing	Do Nothing
SEB-DN-DN	Send Everything in Batches	Do Nothing	Do Nothing
SEB-DN-PS	Send Everything in Batches	Do Nothing	Priority Sampling
SE-DN-PS	Send Everything	Do Nothing	Priority Sampling
CP-DN-IPT	Connectivity and Priority	Do Nothing	Increase Priority Threshold

In Table 7-4, is a summary of the results showing the average performance of each rule over all of the network configurations. The rules are listed in the table in order of increasing average total time real alerts spend in the system before processing. For each performance measure the best performing rules are outlined in bold while the worst performing rules are

shaded in light gray. Tables 7-5 through 7-7 display the results of the aggregated rule performance for each of the network factors: topology, alert rate, and size. Following the summary tables are graphs of each performance measure. For each performance measure graph the rules are listed in the same order as in Table 7-4 to provide consistency between graphs. Each performance measure is discussed following their respective graphs.

Table 7-4: Average Rule Performance Over All Network Configurations and Alert Rates

Rule	Avg. Time to Process Real Alerts (mins)	Real Alerts Processed (%)	Noise Alerts Processed (%)	Avg % of Alerts per Attack (%)	Std Dev of % of Alerts per Attack (%)	Avg Bandwidth (Mbs)
CR-CS-DN	0.01	97.4	92.3	97.8	2.0	0.1600
SAP-CS-DN	0.01	97.5	92.5	98.0	2.0	0.1600
SAP-RS-DN	0.01	97.5	92.5	98.0	2.0	0.1600
SAP-PS-DN	0.03	97.6	92.5	98.0	1.9	0.1600
CR-RS-DN	0.03	97.5	92.3	98.0	2.0	0.1600
CR-PS-PS	0.04	97.1	87.0	97.2	2.5	0.1510
CR-PS-DN	0.09	97.1	87.1	97.2	2.4	0.1512
CP-DN-RS	8.83	95.4	87.8	98.1	4.8	0.1551
CP-DN-ES	15.27	89.8	88.2	96.3	9.8	0.1593
SE-DN-RS	16.42	83.1	92.5	92.6	16.5	0.1688
SEB-DN-RS	19.16	95.7	92.5	97.4	3.8	0.1637
CP-DN-PS	26.39	83.8	88.9	93.0	15.3	0.1637
CP-DN-DN	28.83	83.1	89.2	92.5	16.3	0.1651
SE-DN-IPT	30.87	69.4	92.7	79.0	15.0	0.1645
SEB-DN-IPT	31.84	70.4	92.9	79.5	14.2	0.1653
SE-DN-DN	36.22	80.9	92.5	90.9	19.9	0.1740
SEB-DN-DN	36.52	80.7	92.7	91.0	19.6	0.1734
SEB-DN-PS	36.64	80.6	92.7	91.5	19.0	0.1721
SE-DN-PS	37.70	80.2	92.5	90.8	20.1	0.1733
CP-DN-IPT	37.81	64.3	89.9	74.5	16.5	0.1592

Key

Best performing rules

Worst performing rules

Table 7-5: Average Performance Per Topology

Topology	Avg. Time to Process Real Alerts (mins)	Real Alerts Processed (%)	Noise Alerts Processed (%)	Avg % of Alerts per Attack (%)	Std Dev of % of Alerts per Attack (%)	Avg Bandwidth (Mbs)
1 Layer	16.87	87.3	92.3	93.5	10.4	0.1638
3 Layer	18.85	86.1	90.9	92.4	10.4	0.1634
5 Layer	18.69	87.6	90.0	91.9	10.0	0.1617

Table 7-6: Average Performance Per Alert Rate

Alert Rate	Avg. Time to Process Real Alerts (mins)	Real Alerts Processed (%)	Noise Alerts Processed (%)	Avg % of Alerts per Attack (%)	Std Dev of % of Alerts per Attack (%)	Avg Bandwidth (Mbs)
100%	3.77	96.9	96.6	98.3	2.6	0.1577
115%	32.51	77.0	85.6	86.8	17.9	0.1683

Table 7-7: Average Performance Per Number of Machines

Machines	Avg. Time to Process Real Alerts (mins)	Real Alerts Processed (%)	Noise Alerts Processed (%)	Avg % of Alerts per Attack (%)	Std Dev of % of Alerts per Attack (%)	Avg Bandwidth (Mbs)
250	17.74	87.4	90.9	92.7	10.2	0.1630
1000	18.53	86.5	91.3	92.4	10.3	0.1629

As can be seen in the graph of Figure 7-8, the average time to process alerts had a large range of values, ranging from under 0.1 minutes for the best performing rules to well over 35 minutes for the worst performing rules. The Send Above Priority rules and the Connectivity Revision rules had the best average time performance. In comparison to the Send Above Priority rules and the Connectivity Revision rules all other rules performed relatively poorly for the average time to process real alerts performance measure.

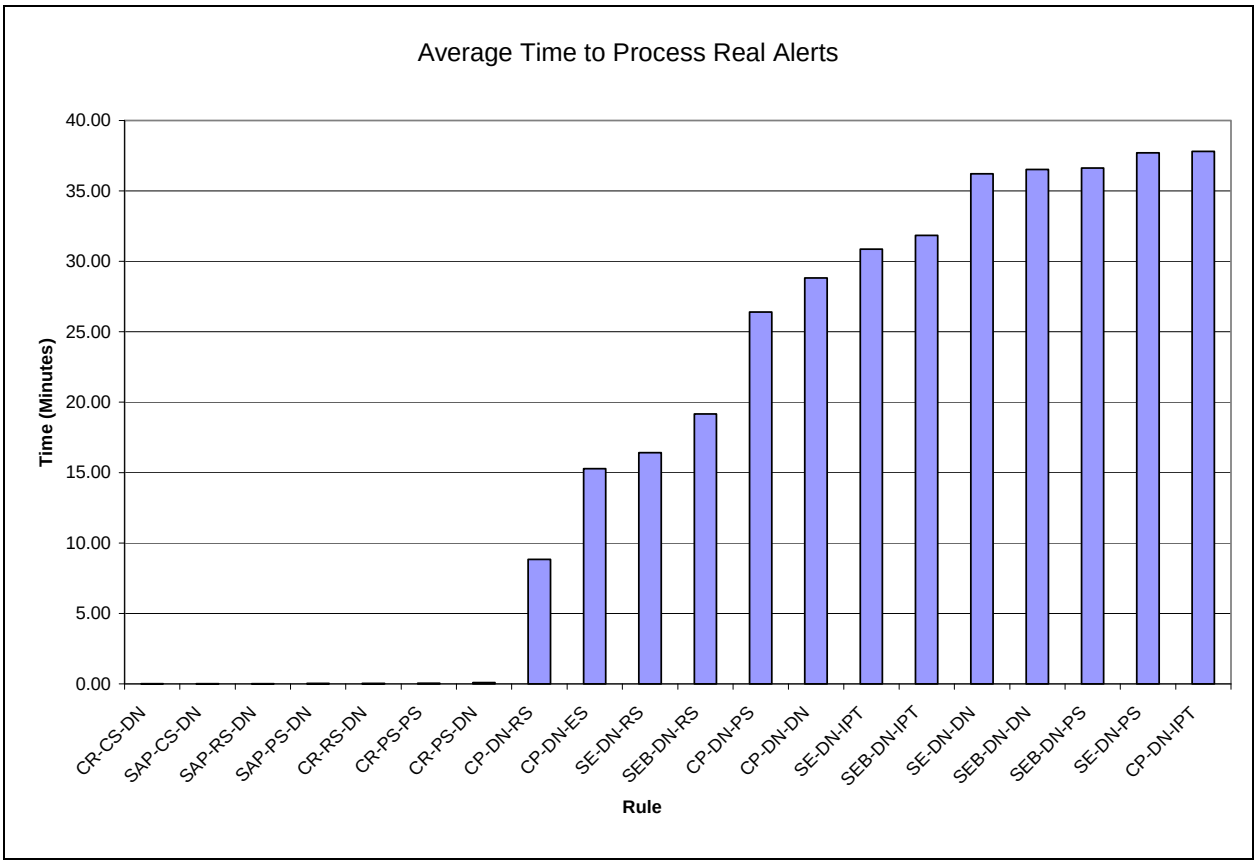

Figure 7-8: Average Time to Process Real Alerts by Rule

Figure 7-9 illustrates that the Connectivity Revision rules and the Send Above Priority rules also performed well for the average percentage of real alerts processed performance measure. The Connectivity Revision and Send Above Priority rules were both able to process over 97% of all real alerts generated. The closest rivals to the top rules were the Send Everything In Batches rule coupled with a random sampling over capacity rule and the Connectivity and Priority rule coupled with a random sampling over capacity rule; the SEB-DN-RS and CP-DN-RS rules processed 95.7% and 95.4% of real alerts respectively. Rules utilizing the increase priority threshold over capacity rule performed poorly for the average percentage of real alerts processed performance measure, with no rule combination using the IPT rule processing more than 70.4% of real alerts.

Observing the percentage of noise alerts processed in conjunction with the percentage of real alerts processed, as in Figure 7-9, provides some perspective to the data. The Connectivity Revision rule with the priority sampling under capacity rule, CR-PS-DN, did a particularly good job of processing a high volume of real alerts, 97.1% on average, and a relatively low volume of noise alerts, 87.1% on average. The rules utilizing the increase priority threshold over capacity rule processed a high volume of noise alerts, while processing a relatively low volume of real alerts compared to the other rules tested.

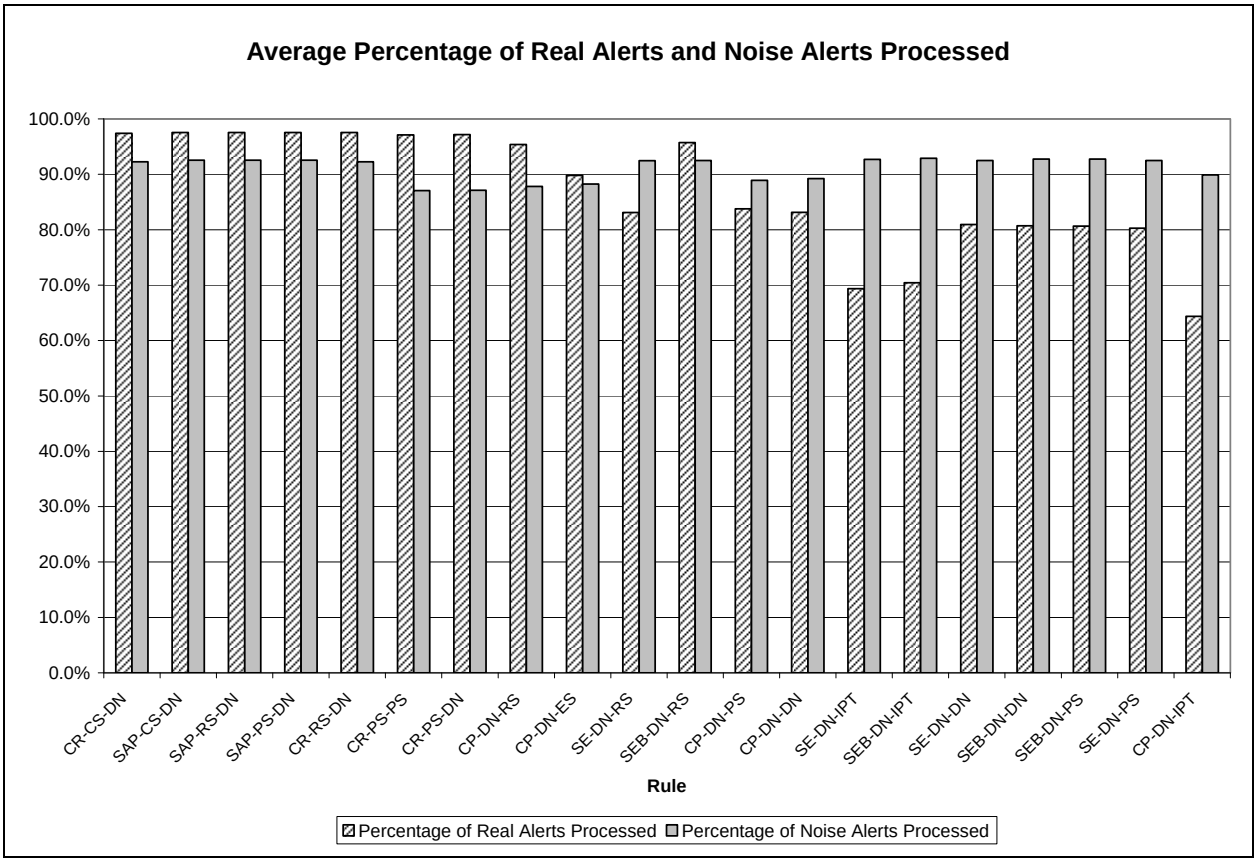

Figure 7-9: Average Percentage of Real and Noise Alerts Processed By Rule

The average percentage of alerts per attack provides a measure of how capable a rule is at providing data to the fusion engine on multiple attacks at a time. As can be seen above in Figure 7-10, the Connectivity Revision and Send Above Priority rules, along with the CP-DN-RS rule

performed very well for this performance measure, with each rule processing over 97% of each attack in the scenario run. In contrast, the increase priority threshold rules processed on average only 80% of each attack.

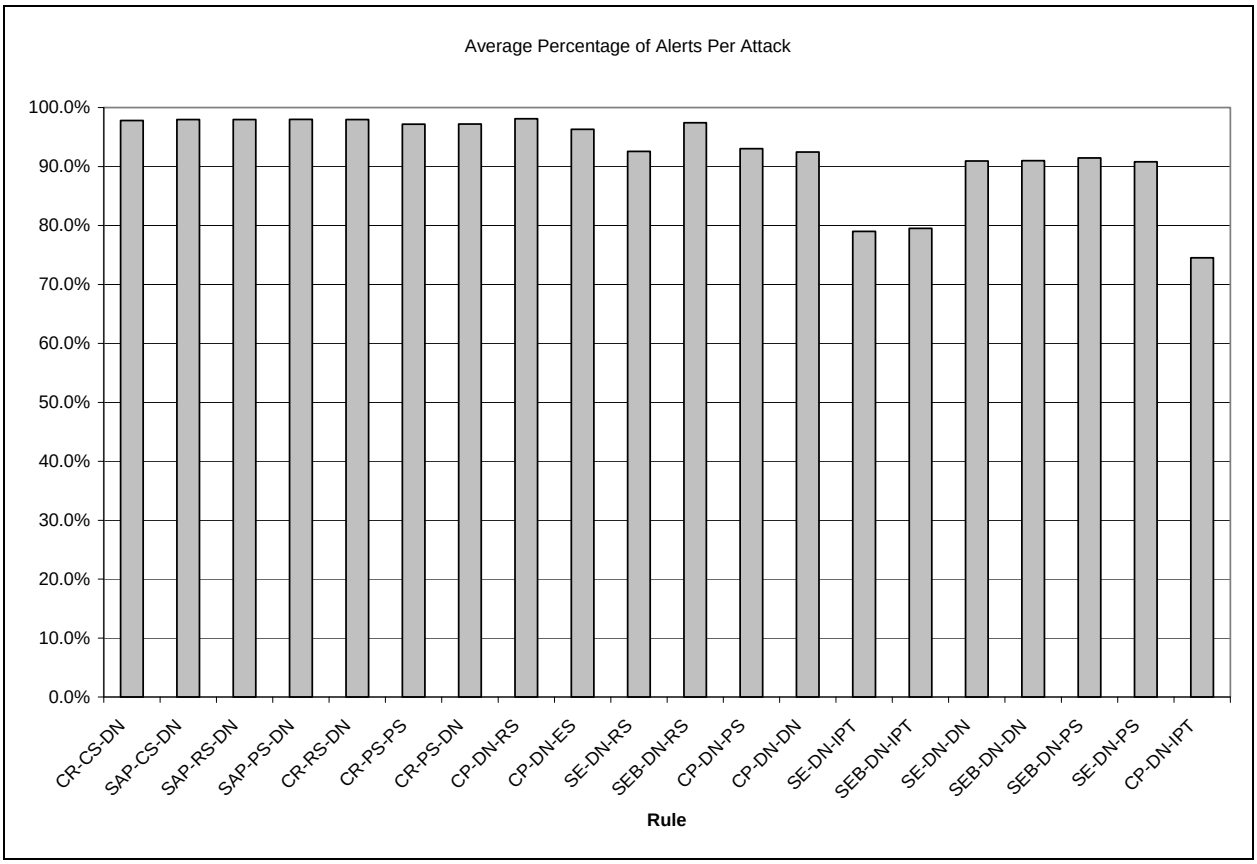

Figure 7-10: Average Percentage of Alerts Per Attack By Rule

While processing a high percentage of each attack is important, another important objective is to process each attack evenly. The ability of a rule to track each attack evenly is indicated by the standard deviation of the percentage of alerts processed per attack. Figure 7-11 above illustrates that the Send Above Priority rules and the Connectivity Revision rules all had much lower average standard deviations than the other rules. The Send Above Priority rule with the priority sampling under capacity rule had the lowest average standard deviation of only 1.8%. The SE-DN-DN and the SEB-DN-DN rules performed relatively poorly with an average

standard deviation of nearly 20%. A high average standard deviation means that for each scenario the SE-DN-DN and the SEB-DN-DN rules tracked some attacks well, while at the same time processing very few alerts from other attacks.

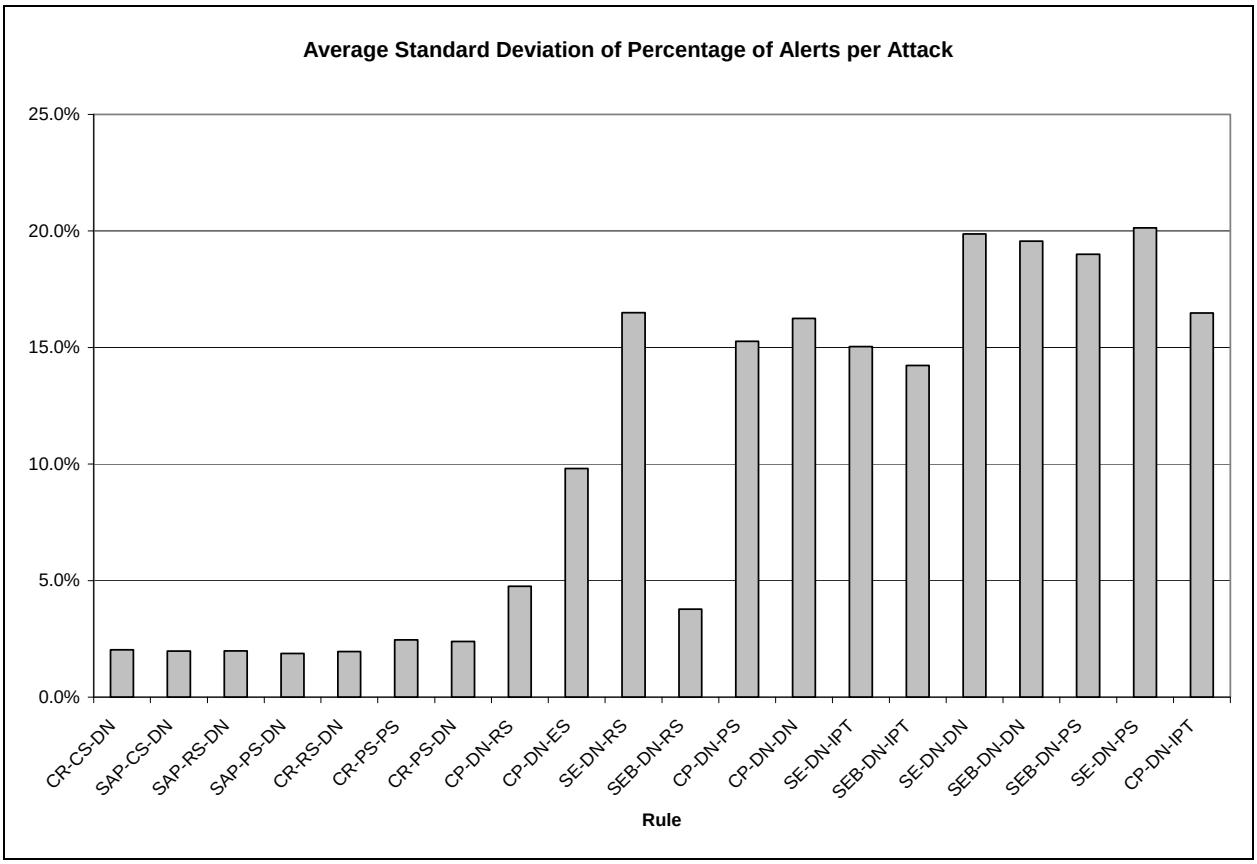

Figure 7-11: Standard Deviation of Percentage of Alerts per Attack By Rule

The amount of bandwidth used is largely related to the amount of data sent to the fusion engine. The CR-PS-DN rule, which typically sent the least amount of data to the fusion engine had the lowest bandwidth use. The rules that sent everything including the SE-DN-DN and the SEB-DN-DN had the highest bandwidth use. The Connectivity Revision and the Send Above Priority rules all had bandwidth use close to 0.16 Mbs, which corresponds to sending 100 alerts per second to the fusion engine.

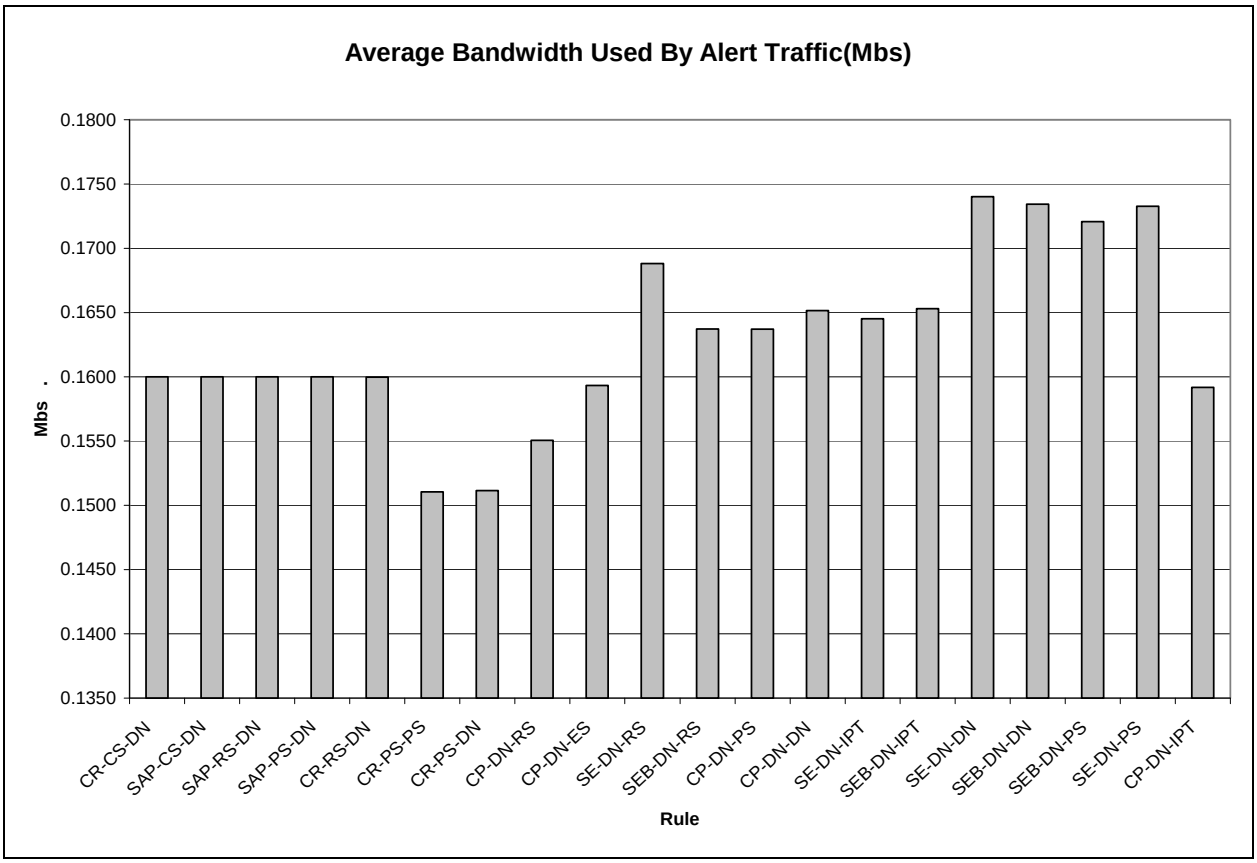

Figure 7-12: Average Bandwidth Used By Alert Traffic By Rule

7.3 Analysis of Results

This section contains a statistical analysis of the results in order to better understand a rule's overall performance. Looking only at the averages of performance measures over all the runs does not necessarily give an accurate picture of rule performance. For this reason, an additional analysis of variance was performed for each performance measure in order to determine if the rule, network topology, alert rate, number of machines in the network, or any two way interactions with the rule had a significant effect on that performance measure. The results of each analysis of variance are presented below followed by a short discussion of each one. A significance level of 0.05 was used to test for significance.

When performing an analysis of variance the data that is analyzed must be normally distributed in order for the ANOVA to be valid. A residual analysis has been performed along with each analysis of variance to ensure that the data is normally distributed. For all of the data sets none of the normality assumptions are violated. A representative residual plot is presented in Figure 7-13, all of the residual plots looked similar to the one presented. The data in the normal probability plot is nearly linear, indicating the data is normal. The histogram of residuals presents a nice bell shape which is another indication that the data is normal. The residuals versus fitted values plot has data evenly distributed on either side of the 0.0 line, indicating that there was no bias in the data at high or low values. Since the data is run on a computer, the order the experiments are run does not make any difference in the outcome of the experiment, therefore the residuals versus the order of the data is not an important plot for this particular experiment.

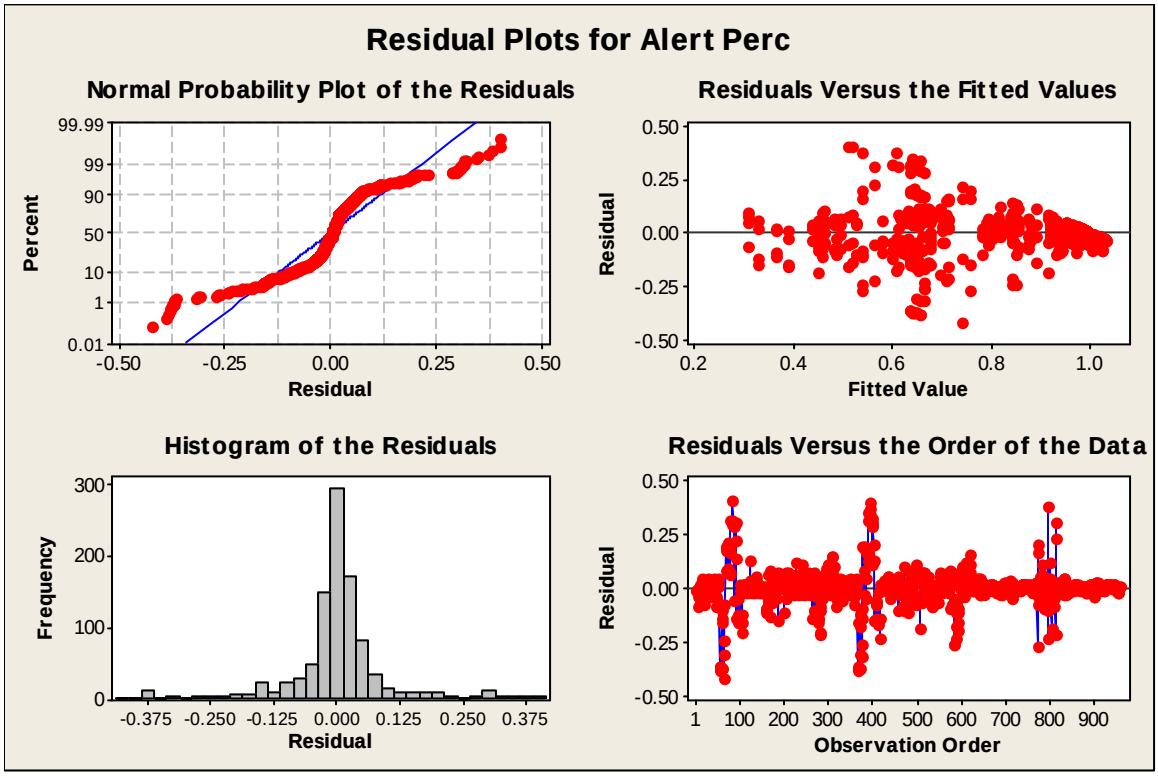

Figure 7-13: Representative Residual Plot

7.3.1 Average Total Time to Process Real Alert

The analysis of variance, shown in Table 7-8, indicates that the rule, topology, and alert rate all have a significant affect on the average total time to process a real alert. There is also a significant interaction effect between the rule and the topology, and the rule and the alert rate. The number of machines in the network does not have a significant effect on the time to process a real alert, nor is there an interaction effect between the rule and the number of machines. This is a significant observation because a network administrator could feel free to increase the number of machines in his network without needing to worry about the affect the extra machines would have on his sensor management strategy.

The alert rate factor has the highest F statistic which suggests that the alert rate plays the largest role in the total time to process a real alert. This observation corresponds closely to common queuing applications. As the rate of entities arriving to a queue increases, the length of the queue, and likewise the waiting time, also increase. As the rate of alerts in the network increases, some of the rules, such as the Send Everything rule, send more alerts to the fusion engine, thereby increasing the length of the fusion engine's queue and causing longer processing times. In contrast to increasing alerts sent to the fusion engine, some rules, such as the Connectivity and Priority rules, cause alerts to wait at sensors longer before sending them to the fusion engine for processing, also causing an increase in the amount of time a real alert spends in the system.

The ANOVA results of Table 7-8 suggest that the rule chosen has a significant affect on the total time to process a real alert. A p-value of 0.00 for the rule factor means that at least one of the rules performs significantly different from the others. A Tukey Test can be performed to determine which rules, if any, perform statistically different from one another. The results of a

Tukey Test on the mean time to process a real alert can be seen in Table 7-8. The Tukey Test indicates that the Send Above Priority and the Connectivity Revision rules perform statistically different from the remaining rules, and also perform significantly better than the remaining rules.

Table 7-8:ANOVA and Tukey Results for Average Total Time to Process Real Alerts

Source	DF	Seq SS	Adj SS	Adj MS	F	P
Rule	19	225798.4	225757	11881.9	336.35	0.000
Topology	2	775.4	777.1	388.6	11.00	0.000
AlertRate	1	198276.9	198306.1	198306.1	5613.53	0.000
Machines	1	149.7	151.1	151.1	4.28	0.039
Rule*Topology	38	6875.9	6791.6	178.7	5.06	0.000
Rule*AlertRate	19	139198.6	139199.9	7326.3	207.39	0.000
Rule*Machines	19	527.1	527.1	27.7	0.79	0.726
Error	860	30380.7	30380.7	35.3		
Total	959	601982.7				

Rule	Time (mins)	Significance
CR-CS-DN	0.01	
SAP-CS-DN	0.01	
SAP-RS-DN	0.01	
CR-RS-DN	0.03	
SAP-PS-DN	0.03	
CR-PS-DN	0.04	
CR-PS-PS	0.04	
CP-DN-RS	8.83	
CP-DN-ES	15.27	
SE-DN-RS	16.42	
SEB-DN-RS	19.16	
CP-DN-PS	26.39	
CP-DN-DN	28.83	
SE-DN-IPT	30.87	
SEB-DN-IPT	31.85	
SE-DN-DN	36.22	
SEB-DN-DN	36.52	
SEB-DN-PS	36.64	
SE-DN-PS	37.70	
CP-DN-IPT	37.81	

An additional analysis of variance was performed on the top performing rules to determine if the alert rate and topology affects these rules, and additionally to determine if there are any significant differences between the top rules. The analysis of variance and Tukey results are shown in Table 7-9. The ANOVA on the Connectivity Revision and Send Above Priority Rules indicates that the rule and network topology have a significant affect on the time to process real alerts for the top performing rules. The alert rate, however, is not a significant factor for the Connectivity Revision and Send Above Priority rules as indicated by the high p-value of 0.356. The Tukey Test shows that the Send Above Priority rules and the CR-CS-DN rule are in the top performing significance group.

Table 7-9: ANOVA and Tukey Results for Top Performing Total Time in System Rules

Source	DF	Seq SS	Adj SS	Adj MS	F	P
Rule	6	0.0548	0.0550	0.0092	8.130	0.000
Topology	2	0.0229	0.0227	0.0114	10.070	0.000
AlertRate	1	0.0009	0.0009	0.0009	0.820	0.367
Machines	1	0.0006	0.0005	0.0005	0.490	0.486
Rule*Topology	12	0.0425	0.0424	0.0035	3.130	0.000
Rule*AlertRate	6	0.0309	0.0309	0.0052	4.570	0.000
Rule*Machines	6	0.0006	0.0006	0.0001	0.080	0.998
Error	301	0.3395	0.3395	0.0011		
Total	335	0.4927				

Rule	Total Time (mins)	Sig
CR-CS-DN	0.01	
SAP-CS-DN	0.01	
SAP-RS-DN	0.01	
SAP-PS-DN	0.03	
CR-RS-DN	0.03	
CR-PS-PS	0.04	
CR-PS-DN	0.04	

The average time to process real alerts performance measure highlights the ability of the Send Above Priority and Connectivity Revision rules to work well under a variety of conditions.

Unlike the remaining rules tested, these rules are not affected by the increase in alert rate and continue to have the same responsive performance even at the higher alert rate. Additionally, although these rules are significantly affected by the topology of the network, their average time to process a real alert is so small to begin with that the increase in time may not be of practical significance.

7.3.2 Percentage of Real Alerts Processed

The analysis of variance results for the percentage of real alerts processed can be seen in Table 7-10. The percentage of real alerts processed is affected by the rule chosen as well as the alert rate. The number of machines in the network and network topology do not have a significant affect on the percentage of alerts processed, however, the interaction between the rule chosen and the network topology does have a significant effect on the percentage of real alerts processed. Additionally, there was a significant interaction effect between the rule chosen and the alert rate of the system, indicated by the 0.000 p-value.

The single largest factor affecting the percentage of real alerts processed was the alert rate. This is not a surprising result because in order to reach the higher alert rate, the amount of noise in the system was increased. For rules that do not distinguish well between noise and real alerts, such as the Send Everything rule, more noise alerts are processed at the higher alert rate, so the percentage of real alerts processed decreases at the higher alert rate.

The Tukey Test in Table 7-10 indicates that rules did perform significantly different from one another. Once again all of the Send Above Priority rules and all of the Connectivity Revision rules are in the top performing group of rules. The Send Above Priority and Connectivity Revision rules are joined by the SEB-DN-RS and CP-DN-RS rules as rules that performed

significantly better than the remaining rules for the percent of real alerts processed performance measure. An additional analysis of variance and Tukey Test is performed on these top performing rules to see if they are also affected by the alert rate, and to see if the top performing rules differ significantly from one another.

Table 7-10: ANOVA and Tukey Results of Percentage of Real Alerts Processed

Source	DF	Seq SS	Adj SS	Adj MS	F	P
Rule	19	10.63	10.63	0.56	64.51	0.00
Topology	2	0.04	0.04	0.02	2.27	0.10
AlertRate	1	9.54	9.54	9.54	1100.11	0.00
Machines	1	0.02	0.02	0.02	2.24	0.14
Rule*Topology	38	1.13	1.12	0.03	3.39	0.00
Rule*AlertRate	19	5.65	5.65	0.30	34.28	0.00
Rule*Machines	19	0.04	0.04	0.00	0.23	1.00
Error	860	7.46	7.46	0.01		
Total	959	34.50				

Rule	Percent	Significance
SAP-PS-DN	97.6%	
SAP-RS-DN	97.6%	
SAP-CS-DN	97.6%	
CR-RS-DN	97.5%	
CR-CS-DN	97.4%	
CR-PS-PS	97.1%	
CR-PS-DN	97.1%	
SEB-DN-RS	95.7%	
CP-DN-RS	95.4%	
CP-DN-ES	89.8%	
CP-DN-PS	83.8%	
CP-DN-DN	83.2%	
SE-DN-RS	83.1%	
SE-DN-DN	80.9%	
SEB-DN-DN	80.7%	
SEB-DN-PS	80.7%	
SE-DN-PS	80.2%	
SEB-DN-IPT	70.5%	
SE-DN-IPT	69.4%	
CP-DN-IPT	64.3%	

The Tukey Test in Table 7-11 did not reveal any significant differences between the Send Above Priority rules, Connectivity Revision rules, the CR-DN-RS rule or the SEB-DN-RS rule. Interestingly, increasing the number of layers in the network had a significant affect on the percentage of real alerts processed. This supports the hypothesis that more layers in a network increase the ability of the connectivity based rules to target and process the real alerts.

The Send Above Priority and Connectivity Revision rules are able to process real alerts in a timely manner. Additionally, these rules process a significantly larger percentage of the real alerts in the system than the other rules tested.

Table 7-11: ANOVA and Tukey Results for Top Performing Rules for Percentage of Real Alerts Processed

Source	DF	Seq SS	Adj SS	Adj MS	F	P
Rule	8	0.0267	0.0269	0.0034	2.41	0.015
Topology	2	0.1322	0.1335	0.0667	47.94	0.000
AlertRate	1	0.1608	0.1606	0.1606	115.35	0.000
Machines	1	0.0000	0.0000	0.0000	0.02	0.894
Rule*Topology	16	0.0278	0.0279	0.0017	1.25	0.225
Rule*AlertRate	8	0.0686	0.0686	0.0086	6.16	0.000
Rule*Machines	8	0.0009	0.0009	0.0001	0.08	1.000
Error	387	0.5387	0.5387	0.0014		
Total	431	0.9558				

Rule	% Real Alerts	Sig
SAP-PS-DN	97.60%	
SAP-RS-DN	97.59%	
SAP-CS-DN	97.58%	
CR-RS-DN	97.54%	
CR-CS-DN	97.44%	
CR-PS-PS	97.12%	
CR-PS-DN	97.08%	
SEB-DN-RS	95.69%	
CP-DN-RS	95.40%	

7.3.3 Percentage of Noise Alerts Processed

The analysis of variance for the percentage of noise alerts processed indicates that all of the factors tested have a significant effect on the percentage of noise alerts processed. This can be seen in Table 7-12 where all the p-values are less than or equal to 0.008. The rule chosen, network topology, alert rate and number of machines in the network all have a significant effect on the percentage of noise alerts processed. The strongest effect was caused by the alert rate in the network. As the alert rate is increased from 100% to 115% of the fusion engine capacity, naturally the fusion engine is unable to process all of the additional alerts in the system, so the percentage of noise alerts processed will fall at the higher alert rate.

The rules that processed significantly less noise include of the CR-PS-PS, CR-PS-DN, and CP-DN-RS rules. If an analyst has a fusion engine that is known to work poorly with lots of noise in the dataset, then choosing one of the three top performing rules could significantly reduce the noise processed by the fusion engine versus the other rules tested. Interestingly, this is the third performance measure in a row that has included the CR-PS-DN and CR-PS-PS rules among the top performing rules.

Table 7-12: ANOVA and Tukey Results for Percentage of Noise Alerts Processed

Source	DF	Seq SS	Adj SS	Adj MS	F	P
Rule	19	0.426	0.428	0.023	155.87	0.000
Topology	2	0.088	0.086	0.043	295.85	0.000
AlertRate	1	2.899	2.897	2.897	20029.18	0.000
Machines	1	0.004	0.004	0.004	26.65	0.000
Rule*Topology	38	0.112	0.112	0.003	20.38	0.000
Rule*AlertRate	19	0.044	0.044	0.002	16.16	0.000
Rule*Machines	19	0.005	0.005	0.000	1.97	0.008
Error	860	0.124	0.124	0.000		
Total	959	3.704				

Rule	Percent of Noise Processed	Significance
CR-PS-PS	87%	
CR-PS-DN	87%	
CP-DN-RS	88%	
CP-DN-ES	88%	
CP-DN-PS	89%	
CP-DN-DN	89%	
CP-DN-IPT	90%	
CR-CS-DN	92%	
CR-RS-DN	92%	
SE-DN-RS	92%	
SE-DN-DN	92%	
SAP-PS-DN	93%	
SAP-CS-DN	93%	
SAP-RS-DN	93%	
SE-DN-PS	93%	
SEB-DN-RS	93%	
SEB-DN-DN	93%	
SE-DN-IPT	93%	
SEB-DN-PS	93%	
SEB-DN-IPT	93%	

7.3.4 Average Alerts Per Attack

The ANOVA results in Table 7-13 indicate that the rule used, network topology and alert rate all have a significant affect on the average percentage of alerts processed per attack. The rule also has a significant interaction with the topology of the network as well as the alert rate. The number of machines in the network does not affect the average percentage of alerts processed per attack. While

The Tukey Test in Table 7-13 demonstrates that there was a large group of rules that performed significantly better than others. The Send Above Priority and Connectivity Revision rules are once again among the top performing rules, and are joined by CP-DN-RS, and SEB-DN-RS. The Send Above Priority and Connectivity Revision rules are able to process alerts in a timely manner, process a high percentage of all real alerts, and are able to process a large percentage of each attack.

Table 7-13: ANOVA and Tukey Results for Average Percentage of Alerts/Attack

Source	DF	Seq SS	Adj SS	Adj MS	F	P
Rule	19	4.542	4.542	0.239	420.00	0.000
Topology	2	0.042	0.042	0.021	36.92	0.000
AlertRate	1	3.193	3.195	3.195	5612.75	0.000
Machines	1	0.002	0.002	0.002	4.14	0.042
Rule*Topology	38	0.450	0.448	0.012	20.71	0.000
Rule*AlertRate	19	2.217	2.217	0.117	204.96	0.000
Rule*Machines	19	0.019	0.019	0.001	1.78	0.021
Error	860	0.490	0.490	0.001		
Total	959	10.954				

Rule	Percent of Alerts/Attack	Significance
CP-DN-RS	98%	
SAP-PS-DN	98%	
SAP-CS-DN	98%	
SAP-RS-DN	98%	
CR-RS-DN	98%	
CR-CS-DN	98%	
SEB-DN-RS	98%	
CR-PS-PS	97%	
CR-PS-DN	97%	
CP-DN-ES	96%	
CP-DN-PS	93%	
SE-DN-RS	93%	
CP-DN-DN	93%	
SEB-DN-PS	92%	
SEB-DN-DN	91%	
SE-DN-DN	91%	
SE-DN-PS	91%	
SEB-DN-IPT	80%	
SE-DN-IPT	79%	
CP-DN-IPT	75%	

7.3.5 Standard Deviation of Average Alerts Per Attack

The standard deviation of the alerts per attack provides a measure of the consistency with which a rule tracks multiple attacks. A low standard deviation indicates that a rule tracks each attack in the scenario with the same thoroughness. The ANOVA results for this performance measure can be seen in Table 7-14. The rule chosen as well as the alert rate of the system have a

significant affect on the standard deviation of alerts processed per attack. There is also a significant interaction between the rule and the network topology as well as the rule and the alert rate. The number of machines in the network does not affect the standard deviation of alerts per attack.

Table 7-14: ANOVA and Tukey Results for Standard Deviation of Alerts Per Attack

Source	DF	Seq SS	Adj SS	Adj MS	F	P
Rule	19	5.130	5.130	0.270	280.27	0.000
Topology	2	0.004	0.004	0.002	1.99	0.137
AlertRate	1	5.642	5.642	5.642	5856.68	0.000
Machines	1	0.000	0.000	0.000	0.36	0.548
Rule*Topology	38	0.201	0.200	0.005	5.46	0.000
Rule*AlertRate	19	4.005	4.005	0.211	218.81	0.000
Rule*Machines	19	0.009	0.009	0.000	0.48	0.971
Error	860	0.828	0.828	0.001		
Total	959	15.820				

Rule	Stdev of Percent of Alert/Attack	Significance
SAP-PS-DN	2%	
CR-RS-DN	2%	
SAP-CS-DN	2%	
SAP-RS-DN	2%	
CR-CS-DN	2%	
CR-PS-DN	2%	
CR-PS-PS	3%	
SEB-DN-RS	4%	
CP-DN-RS	5%	
CP-DN-ES	10%	
SEB-DN-IPT	14%	
SE-DN-IPT	15%	
CP-DN-PS	15%	
CP-DN-DN	16%	
CP-DN-IPT	16%	
SE-DN-RS	17%	
SEB-DN-PS	19%	
SEB-DN-DN	20%	
SE-DN-DN	20%	
SE-DN-PS	20%	

The Tukey Test for the rules in Table 7-14 indicates that the Send Above Priority, Connectivity Revision, and the SEB-DN-RS rules performed significantly better than the other rules tested. The top performing rules are subjected to their own analysis of variance and Tukey Tests to determine if any of the top performing rules differ significantly from one another, and to see if the alert rate has as profound an effect on the top performers. The results of the ANOVA on the top performing rules can be seen in Table 7-15.

Table 7-15: ANOVA and Tukey Results for the Top Performing Rules of the Standard Deviation of Alerts Processed Per Attack Performance Measure

Source	DF	Seq SS	Adj SS	Adj MS	F	P
Rule	7	0.0133	0.0133	0.0019	16.06	0.000
Topology	2	0.0062	0.0064	0.0032	27.05	0.000
AlertRate	1	0.0751	0.0749	0.0749	633.06	0.000
Machines	1	0.0000	0.0000	0.0000	0.00	0.998
Rule*Topology	14	0.0027	0.0027	0.0002	1.64	0.068
Rule*AlertRate	7	0.0252	0.0252	0.0036	30.44	0.000
Rule*Machines	7	0.0001	0.0001	0.0000	0.10	0.998
Error	344	0.0407	0.0407	0.0001		
Total	383	0.1633				

Rule	Stdev of Percent of Alert/Attack	Sig
SAP-PS-DN	1.86%	
CR-RS-DN	1.96%	
SAP-CS-DN	1.99%	
SAP-RS-DN	2.00%	
CR-CS-DN	2.05%	
CR-PS-DN	2.41%	
CR-PS-PS	2.53%	
SEB-DN-RS	3.72%	

The ANOVA on the top performing rules indicates that the network topology and the alert rate are still important factors. The Tukey Test on the top performing rules themselves indicates that the SEB-DN-RS rule performs significantly worse than the other top performing rules. There were no significant differences between the Send Above Priority rules, and the

Connectivity Revision rules. These rules are able to track multiple attacks with the same accuracy. All of these rules appeared among the top performing rules for the average alerts processed per attack performance measure, which suggests that these rules are not only good at tracking attacks evenly, but are also good at tracking a large proportion of each attack.

7.3.6 Average Bandwidth Used

The ANOVA and Tukey results for average bandwidth usage can be seen below in Table 7-16. The rule, topology, alert rate, and number of machines all have significant affects on the amount of bandwidth used. Additionally, there is a significant interaction between the rule and the topology as well as between the rule and the alert rate. The Tukey test on the rules indicates that the CR-PS-PS, CR-PS-DN, and CP-DN-RS rules performed significantly better than the other rules tested. These three rules used less bandwidth, which also means they sent fewer alerts to the fusion engine. All three of these rules were also among the rules that sent significantly less noise to the fusion engine, so the reduced bandwidth usage may coincide with the reduction in noise sent to the fusion engine by the rules.

A reduction in bandwidth is only valuable if the rule is still processing a high percentage of real alerts in a timely manner. The CR-PS-PS and CR-PS-DN rules were among the top performing rules for the average time to process a real alert metric as well as the percentage of real alerts processed, average percentage of alerts per attack, and standard deviation of alerts per attack metrics. At stated earlier the CR-PS-PS and CR-PS-DN rules were also among the rules that successfully reduced noise sent to the fusion engine.

Table 7-16: ANOVA and Tukey Results for Average Bandwidth Usage

Source	DF	Seq SS	Adj SS	Adj MS	F	P
Rule	19	0.042	0.043	0.002	309.19	0.000
Topology	2	0.001	0.001	0.000	52.00	0.000
AlertRate	1	0.027	0.027	0.027	3719.08	0.000
Machines	1	0.000	0.000	0.000	0.11	0.739
Rule*Topology	38	0.004	0.004	0.000	15.73	0.000
Rule*AlertRate	19	0.022	0.022	0.001	156.69	0.000
Rule*Machines	19	0.001	0.001	0.000	4.82	0.000
Error	860	0.006	0.006	0.000		
Total	959	0.103				

Rule	Bandwidth (Mbs)	Significance
CR-PS-PS	0.150	
CR-PS-DN	0.151	
CP-DN-RS	0.155	
CP-DN-ES	0.159	
CP-DN-IPT	0.160	
CR-CS-DN	0.160	
CR-RS-DN	0.160	
SAP-CS-DN	0.160	
SAP-PS-DN	0.160	
SAP-RS-DN	0.160	
SEB-DN-RS	0.162	
CP-DN-PS	0.163	
SE-DN-IPT	0.164	
SEB-DN-IPT	0.165	
CP-DN-DN	0.166	
SE-DN-RS	0.168	
SEB-DN-PS	0.171	
SE-DN-PS	0.172	
SEB-DN-DN	0.173	
SE-DN-DN	0.173	

7.4 Discussion

While reviewing each performance measure individually is important, all the performance measures analyzed together can be used to gain an understanding of which rules, if any, have significantly better performance than others. Table 7-17 displays each performance measure and the performance significance groups that each rule fell into. Rules with the lightest

gray areas beneath a performance measure are in the top performance group for that measure. Black shading means the rule was not in the top three performance groups.

Table 7-17 clearly shows that the Connectivity Revision and Send Above Priority Rules are the best performing rules across many of the different performance measures. The Send Above Priority Rules and the Connectivity Revision rules are in the top performance group for the percentage of real alerts processed, the average percentage of alerts processed per attack, and the standard deviation of alerts processed per attack. In addition to these key measures the CR-PS-DN and CR-PS-PS rules were also in the top performance group for the amount of noise sent to the fusion engine and the bandwidth use in the network. CR-PS-PS and CR-PS-DN rules are only rules that significantly reduced the amount of noise sent to the fusion engine, while still performing excellently for all other performance measures. The CR-PS-PS and CR-PS-DN rules were in the second performance group for the total time a real alert spends in the system, taking an average of 1.35 seconds longer to process a real alert than the Send Above Priority Rules. The Send Above Priority Rules have the fastest total time in system performance, processing real alerts in an average of 1.04 seconds. Since the Send Above Priority rules did not reduce the amount of noise sent to the fusion engine these rules effectively processed the most data in the least amount of time.

Table 7-17: Summary of Rule Performance Over All Performance Measures

	Avg Time to Process Real Alerts	Real Alerts Processed	Noise Alerts Processed	Avg % of Alerts per Attack	Std Dev of % of Alerts per Attack	Avg Bandwidth
CR-PS-DN						
CR-PS-PS						
SAP-RS-DN						
SAP-CS-DN						
SAP-PS-DN						
CR-CS-DN						
CR-RS-DN						
CP-DN-RS						
SEB-DN-RS						
CP-DN-ES						
CP-DN-DN						
CP-DN-PS						
SE-DN-RS						
CP-DN-IPT						
SEB-DN-PS						
SE-DN-DN						
SE-DN-PS						
SEB-DN-DN						
SE-DN-IPT						

Key

	Top Performance Group
	Second Performance Group
	Third Performance Group
	Fourth or Higher Performance Group

This experiment has not highlighted a one best rule for sensor management, but rather has shown there are several considerations to take when choosing a sensor management strategy. If bandwidth use and/or noise reduction are important concerns for a particular cyber security fusion system then rules such as the CR-PS-DN and CR-PS-PS rules have excellent performance across all performance measures, and reduce the amount of noise sent to the fusion engine which has the additional benefit of reducing bandwidth use. If bandwidth and noise reduction are not priorities for a cyber security system then the Send Above Priority rules have shown an ability to

process a high volume of real alerts extremely quickly, while still processing high volumes of noise alerts.

Another consideration when choosing between rules is whether the 1.35 second significant difference between the Send Above Priority rules and the Connectivity Revision rules is of practical significance. An analyst would have to ask himself, “Would my network security be impacted if I learned about attacks 1.35 seconds later?” The answer to this question depends on a system’s typical reaction rate to attacks, and any intrusion prevention devices that the network may have that could stop an attack in progress.

Another consideration when deciding to use a sensor management rule is the ease with which a rule can be implemented into a real system. The Connectivity Revision rules have been shown to work well, but these rules rely heavily on the ability of a fusion engine to produce correct attack tracks. In this experiment the fusion engine was 100% accurate at identifying the attack tracks, but in a real system a fusion engine may generate false attack tracks, or miss attack tracks completely which could impact the performance of the connectivity based rules. In addition to fusion engine input, the Connectivity Revision rules require network topology information to be accurate. Providing the Connectivity Revision rules with accurate attack tracks from the fusion engine and network topology information may present several implementation challenges. Conversely, the Send Above Priority rules do not require fusion engine input or network topology information. The Send Above Priority rules work well regardless of how accurate the fusion engine is at producing attack tracks and how current network topology information for the network is.

There were some surprising results in the performance of two of the sub rules, particularly the performance of the priority sampling rule, and the poor performance of the

increase priority threshold rule. The ability of the CR-PS-DN and CR-PS-PS rules to reduce the amount of noise sent to the fusion engine while still performing well for all other performance measures deserves some additional attention. The priority sampling rule employed has the benefit of reducing the amount of noise sent to the fusion engine. This is most likely caused by the polling mechanism the priority sampling rule employs. Recall that the priority sampling rule polls all sensors for one alert, records the priority of the alert received, and then re-polls the sensor that sent the alert with the highest priority. The initial polling of all sensors is done the first time the rule is called, and then again every ten times the rule is used, or roughly every ten seconds. If a sensor has no alerts to send at the initial polling, the sensor will not be polled again for ten seconds. During this time alerts may arrive at the sensor that will have to wait until another polling of all sensors before they have a chance to be sent. The alerts available from the other sensors that had alerts at the initial polling may not be enough to fill the fusion engine capacity, so less alerts are sent to the fusion engine during that time period.

The increase priority threshold rule's performed extremely poorly even though the rule was designed to improve performance. Reasons for the increase priority rule's poor performance can be attributed to at least two factors including the interval that the queue at the fusion engine is checked and the distribution of alerts over the different priorities. The one second interval between checking the length of the fusion engine queue sometimes allowes thousands of alerts to flood the fusion engine. During a one second interval there is a possibility that the fusion engine could go from having zero alerts in queue to 6,000 alerts, the maximum allowed by bandwidth in once second. This onslaught of alerts does not allow the increase priority threshold rule any chance to increase the priority threshold in hopes of reducing the number of alerts sent to the fusion engine. By the time the queue is checked, the queue length is already passed the final

threshold level. One way to combat this problem is to check the fusion engine queue length more often than every second, and in a best case scenario check the length of the fusion engine queue each time an alert is added to the queue. Decreasing the time interval between checking the fusion engine queue will allow the increase priority threshold rule a chance to increase the priority threshold in order to stop the growth of the queue. Decreasing the time interval may increase the processing requirements of this rule which may have a negative impact on the system as a whole.

A second factor in the poor performance of the increase priority threshold rule is the distribution of priorities of alerts in the system. The overwhelming majority of alerts were priority level 0 alerts, or reconnaissance alerts. Having a large percentage of priority level zero alerts means that when the queue length at the fusion engine reduces and the increase priority threshold rule is able to reduce the threshold level back down to a priority level of zero, a huge volume of alerts becomes available and is sent to the fusion engine in the following one second interval. This causes the priority threshold rule to increase the threshold to the highest level immediately after just reducing the threshold. This problem could be combated in two ways. Firstly, the sensors could perform some archiving function of low priority alerts that are not sent after a period of time, so that the fusion engine is not flooded with the old low priority alerts once the priority level is reduced. Secondly, a new priority scheme could be developed that has more levels and more evenly distributes the alert types across all the levels.

8 CONCLUSIONS & RECOMMENDATIONS FOR FUTURE RESEARCH

This research has demonstrated the importance of sensor management strategies to a properly functioning cyber information fusion system. Unique event based sensor management rules have been developed, a simulation environment has been created to test the sensor management rules, and an experiment has been designed and performed. The experiment highlights several rules robust performance as well as indicates factors that are important to the performance of sensor management rules.

In general, the sensor management strategies that perform the best are those that keep the queue at the fusion engine short. The Connectivity Revision and Send Above Priority rules do an excellent job of feeding the fusion engine exactly what the engine can process, thereby keeping the queue short and the system as a whole very responsive.

Several factors may influence an analyst's decision to implement a particular rule. These factors may include whether or not the analyst's fusion engine is sensitive to high volumes of noise, whether the network has tight bandwidth restrictions that must be met, and how fast the typical reaction rate to an attack in progress is. If the analyst is extremely concerned about the volume of noise at the fusion engine or bandwidth use, he may want to consider the Connectivity Revision-Priority Sampling rules. These rules send significantly less noise to the fusion engine, while still processing the same amount of real alerts as the other top performing rules. The Connectivity – Priority Sampling rules decrease noise sent and bandwidth use, but the average time taken to process a real alert is slightly higher than the Send Above Priority Rules. If the time to process alerts is of high importance, or bandwidth use and the amount of noise sent to the fusion engine are not big concerns, then the Send Above Priority rules may be a good choice.

These rules were among the best performing rules for their ability to process alerts quickly, process a high percentage of real alerts, and process alerts from multiple attacks evenly.

In addition to identifying sensor management strategies that worked well, this research also identifies network factors that affect the performance of sensor management rules. The network topology has a large affect on the connectivity based sensor management rules. As the number of layers in the network increases, the connectivity based sensor management rules see an increase in performance. Specifically, as the number of layers in the network increases the connectivity based rules improve their ability to send less noise to the fusion engine. The alert rate of the network also has a large affect on the performance of sensor management rules. At the low alert rate many sensor management rules perform well, but at higher alert rates the rules that pay more attention to the constraints in the network have better performance. In future research including both the network topology and alert rate would be important factors to include in an experiment. The number of machines in the network does not have a significant affect on the performance of rules, and may be left out of future experiments.

There is a considerable amount of future research that could be done on this topic. Additional research could include changing the simulation environment, including different experimental factors, incorporating a more accurate fusion engine into the simulation model, and the developing more rules.

In future experiments a larger number of scenarios could be evaluated along with more replications of each scenario. In this experiment one scenario setup is used in an attempt to capture a variety of attack conditions that may occur during any given scenario. In future experiments, several different scenarios could be tried on one network configuration to ensure that rules perform consistently for several different attack conditions. Scenarios could include a

higher percentage of real alerts to investigate the performance of rules when the rate of real alerts exceeds the fusion engine capacity. Additionally, including more scenario replications could help to differentiate the performance of the rules.

In this experiment the fusion queue length thresholds are set to 60 seconds for the max queue and 6 seconds for the threshold. These threshold queue levels are kept constant for all the rules tested. The performance of the over capacity rules can likely be improved by adjusting these threshold levels to other values.

The interval set to monitor the fusion engine queue length is set to once every second. This interval could be made smaller, so that the fusion engine queue has less of a chance to grow. The interval between monitoring the fusion engine queue length could be changed to check the queue every time an alert is added to the fusion engine queue. Dynamically checking the fusion engine queue could increase the processing requirements of a sensor management system, but at the same time checking the queue more often could drastically improve the performance of many of the over capacity rules.

The fusion engine model in this research processed all attacks with 100% accuracy. In a real information fusion system a fusion engine may create attack tracks for attacks that do not exist, or conversely miss attack tracks for attacks that do exist. The connectivity based rules of this research rely heavily on the performance of the fusion engine in order to control the settings of sensors. In future research, a fusion engine could be modeled more accurately to include the creation of false attack tracks, or to miss attacks in order to study the affect a suboptimal fusion engine has on the connectivity based rules.

Finally, changes could be made to the cyber simulation environment. An interface could be developed to allow a new user to easily create and test new rules without needing to know

how to program in Java. The sensor management package could also be streamlined so that a new user could easily code new rules without having to make updates to several classes and functions.

REFERENCES

Bass, T. (2000) Intrusion detection systems and multisensor data fusion. *Communications of the ACM, 43(4),* 99-105.

Bejtlich, R. (2005) *The tao of network security monitoring:Beyond intrusion detection*. Boston: Pearson Education Inc.

Costantini, K. (2007) Unpublished master thesis, Rochester Institute of Technology, Rochester.

Crothers, T. (2003) *Implementing intrusion detection systems*. Indianapolis: Wiley Publishing, Inc.

Holsopple, J., Yang, S. J., & Sudit, M. (2006) TANDI: Threat assessment of network data and information. *Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications 2006, Apr 19-20 2006, 6242*. from http://dx.doi.org/10.1117/12.665288

Kang, M. H., & Mayfield, T. (2003) A cyber-event correlation framework and metrics. *PROCEEDINGS OF SPIE SPIE - the International Society for Optical Engineering: System Diagnosis and Prognosis: Security and Condition Monitoring Issues III, Apr 21 2003, 5107* 72-82. from http://dx.doi.org/10.1117/12.488029

Kistner, J. (2006) *Cyber attack simulation and information fusion process refinement optimization models for cyber security*. Unpublished master thesis, Rochester Institute of Technology, Rochester.

Krishnamurthy, V. (2002) Algorithms for optimal scheduling and management of hidden markov model sensors. *IEEE Transactions on Signal Processing, 50*(6), 1382-97.

Kuhl, M., Kistner, J., Costantini, K., & Sudit, M. (2007) Cyber attack modeling and simulation for network security analysts. *Proceedings of the Winter Simulation Conference, Dec 2007,* to appear.

Maran Graphics Inc. (1997) *Teach yourself networking visually*. Foster City, CA: IDG Books Worldwide.

Mullen, T., Avasarala, V., & Hall, D. L. (2006) Customer-driven sensor management. *IEEE Intelligent Systems, 21*(2), 41-49.

Nicholson, D., & Leung, V. (2004) Managing a distributed data fusion network. *Signal Processing, Sensor Fusion, and Target Recognition XIII, Apr 12-14 2004, 5429,* 129-137. from http://dx.doi.org/10.1117/12.543612

Sabata, B., & Ornes, C. (2006) Multi-source evidence fusion for cyber-situation assessment. *Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications 2006, Apr 19-20 2006, 6242,* 624201. from http://dx.doi.org/10.1117/12.663436

Sciacca, L. J., & Evans, R. J. (2002) Cooperative sensor networks with bandwidth constraints. *Battlespace Digitization and Network - Centric Warfare II, Apr 3-5 2002, 4741,* 192-201. from http://dx.doi.org/10.1117/12.478712

Sudit, M., Stotz, A., & Holender, M. (2005) Situational awareness of a coordinated cyber attack1. *Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security 2005, Mar 28-29 2005, 5812,* 114-129. from http://dx.doi.org/10.1117/12.606980

Viswanath, A., Mullen, T., Hall, D., & Garga, A. (2005) MASM: A market architecture for sensor management in distributed sensor networks. *Multisensor, Multisource Information Fusion: Architecture, Algorithms, and Applications 2005, Mar 30-31 2005, 5813,* 281-289. from http://dx.doi.org/10.1117/12.604074

Wang, X., Evans, R., & Legg, J. (2004) Distributed sensor fusion with network constraints. *Signal Processing, Sensor Fusion, and Target Recognition XIII, Apr 12-14 2004, 5429,* 150-161. from http://dx.doi.org/10.1117/12.541954

Yang, J. & Sikdar, B. (2005) Lightweight target tracking protocol using ad-hoc sensor network. *Institute of Electrical and Electronics Engineers Inc., 2005, 61,* 2850-2854.

Zhang, J., Premaratne, K., & Bauer, P. H. (2002) Resource allocation and congestion control in distributed sensor networks - a network calculus approach. *MTNS-02: 15th International Symposium on Mathematical Theory of Networks and Systems, 12-16 Aug. 2002,* 1-10.

APPENDECIES

Appendix I: Detailed Experimental Results

This appendix contains the average rule performance of each of the original 58 rule combinations tested. The results are presented in tables grouped by the main rule. There are tables for the Send Everything Rules, Connectivity and Priority Rules, Send Above Priority Rules, Send Everything in Batches Rules, and Connectivity Revision Rules.

Table I-1: Average Performance of the Send Everything Rules over Experimental Factors

Rule	Avg. Time to Process Real Alerts (mins)	Real Alerts Processed (%)	Noise Alerts Processed (%)	Avg. % of Alerts per Attack (%)	Std Dev of % of Alerts per Attack (%)	Avg. Bandwidth (Mbs)
SE-DN-RS	16.420	83.1	92.5	92.6	16.5	0.169
SE-DN-IPT	30.872	69.4	92.7	79.0	15.0	0.165
SE-DN-DN	36.218	80.9	92.5	90.9	19.9	0.174
SE-DN-PS	37.701	80.2	92.5	90.8	20.1	0.173
Topology						
1 Layer	28.154	81.3	93.0	90.8	16.4	0.170
3 Layer	30.970	75.7	92.3	87.4	18.8	0.170
5 Layer	31.783	78.2	92.2	86.8	18.4	0.171
Alert Rate						
100%	6.762	95.3	98.5	97.6	3.7	0.162
115%	53.843	61.6	86.6	79.0	32.1	0.179
Machines						
250	29.593	79.3	92.6	88.6	17.8	0.170
1000	31.012	77.6	92.5	88.1	18.0	0.170

Table I-2: Average Performance of the Connectivity and Priority Rules over Experimental Factors

Rule	Avg. Time to Process Real Alerts (mins)	Real Alerts Processed (%)	Noise Alerts Processed (%)	Avg. % of Alerts per Attack (%)	Std Dev of % of Alerts per Attack (%)	Avg. Bandwidth (Mbs)
CP-CS-RS	9.109	94.8	92.3	98.3	4.5	0.163
CP-DN-RS	8.831	95.4	87.8	98.1	4.8	0.155
CP-PS-RS	9.029	95.5	91.1	98.4	4.1	0.161
CP-RS-RS	8.354	95.6	92.3	98.4	3.9	0.162
CP-CS-ES	15.284	89.8	92.4	96.6	9.2	0.166
CP-DN-ES	15.272	89.8	88.2	96.3	9.8	0.159
CP-PS-ES	15.368	89.6	91.3	96.5	9.7	0.164
CP-RS-ES	15.318	90.0	92.4	96.7	9.1	0.166
CP-CS-PS	29.825	82.7	92.5	92.8	16.5	0.171
CP-DN-PS	26.392	83.8	88.9	93.0	15.3	0.164
CP-PS-PS	26.789	72.8	91.7	81.8	15.0	0.168
CP-RS-PS	31.668	82.5	92.5	92.3	17.3	0.172
CP-CS-DN	29.122	83.4	92.5	92.7	15.9	0.171
CP-DN-DN	28.827	83.1	89.2	92.5	16.3	0.165
CP-PS-DN	29.133	83.3	91.6	92.6	16.1	0.169
CP-RS-DN	29.180	83.3	92.5	92.7	16.0	0.171
CP-CS-IPT	33.023	64.7	92.7	75.1	16.3	0.164
CP-DN-IPT	37.811	64.3	89.9	74.5	16.5	0.159
CP-PS-IPT	36.433	65.5	91.9	75.4	16.4	0.163
CP-RS-IPT	36.180	64.7	92.7	75.0	16.6	0.165
Topology						
1 Layer	24.042	82.9	92.0	91.6	14.0	0.166
3 Layer	24.033	81.0	91.1	90.2	12.4	0.165
5 Layer	22.567	84.3	90.9	89.7	11.0	0.164
Alert Rate						
100%	3.518	97.2	96.8	98.3	2.8	0.158
115%	43.577	68.2	85.9	82.7	22.1	0.172
Machines						
250	23.296	83.2	91.2	12.3	90.8	0.165
1000	23.799	82.2	91.4	12.6	90.1	0.165

Table I-3: Average Performance of the Send Above Priority Rules over Experimental Factors

Rule	Avg. Time to Process Real Alerts (mins)	Real Alerts Processed (%)	Noise Alerts Processed (%)	Avg. % of Alerts per Attack (%)	Std Dev of % of Alerts per Attack (%)	Avg. Bandwidth (Mbs)
SAP-CS-DN	0.013	97.5	92.5	98.0	2.0	0.160
SAP-PS-DN	0.032	97.6	92.5	98.0	1.9	0.160
SAP-RS-DN	0.013	97.5	92.5	98.0	2.0	0.160
Topology						
1 Layer	0.019	95.5	93.7	97.2	2.5	0.160
3 Layer	0.019	98.7	92.0	98.3	1.8	0.160
5 Layer	0.019	98.4	91.9	98.5	1.5	0.160
Alert Rate						
100%	0.020	98.7	98.4	99.5	0.9	0.160
115%	0.018	96.4	86.7	96.5	3.0	0.160
Machines						
250	0.019	97.4	92.3	97.9	2.0	0.160
1000	0.020	97.7	92.7	98.1	1.9	0.160

Table I-4: Average Performance of the Send Everything In Batches Rules over Experimental Factors

Rule	Avg. Time to Process Real Alerts (mins)	Real Alerts Processed (%)	Noise Alerts Processed (%)	Avg. % of Alerts per Attack (%)	Std Dev of % of Alerts per Attack (%)	Avg. Bandwidth (Mbs)
SEB-CS-RS	16.772	87.4	92.4	95.0	11.5	0.170
SEB-DN-RS	19.163	95.7	92.5	97.4	3.8	0.164
SEB-PS-RS	16.964	85.3	92.4	94.1	13.9	0.170
SEB-RS-RS	17.012	87.4	92.4	94.9	11.6	0.170
SEB-CS-IPT	32.290	69.7	92.7	78.9	14.9	0.164
SEB-DN-IPT	31.844	70.4	92.9	79.5	14.2	0.165
SEB-PS-IPT	30.209	70.5	92.7	79.2	14.4	0.165
SEB-RS-IPT	30.585	70.4	92.7	79.0	14.5	0.164
SEB-CS-PS	37.734	80.5	92.5	91.1	19.8	0.173
SEB-DN-PS	36.637	80.6	92.7	91.5	19.0	0.172
SEB-PS-PS	29.324	74.2	92.6	82.5	14.3	0.170
SEB-RS-PS	37.242	80.6	92.5	91.1	19.8	0.173
SEB-CS-DN	37.542	80.7	92.5	90.7	20.2	0.174
SEB-DN-DN	36.524	80.7	92.7	91.0	19.6	0.173
SEB-PS-DN	37.545	80.7	92.5	90.7	20.2	0.174
SEB-RS-DN	37.543	80.7	92.5	90.7	20.2	0.174
Topology						
1 Layer	27.381	82.3	93.2	90.9	15.0	0.169
3 Layer	31.280	76.9	92.3	87.6	16.6	0.170
5 Layer	32.263	80.0	92.2	87.2	15.6	0.170
Alert Rate						
100%	7.115	95.1	98.5	97.5	3.8	0.161
115%	53.501	64.3	86.7	79.7	27.6	0.178
Machines						
250	29.901	80.8	92.5	89.1	15.6	0.170
1000	30.715	78.6	92.6	88.0	15.9	0.169

Table I-5: Average Performance of the Connectivity Revision Rules over Experimental Factors

Rule	Avg. Time to Process Real Alerts (mins)	Real Alerts Processed (%)	Noise Alerts Processed (%)	Avg. % of Alerts per Attack (%)	Std Dev of % of Alerts per Attack (%)	Avg. Bandwidth (Mbs)
CR-CS-DN	0.013	97.4	92.3	97.8	2.0	0.160
CR-CS-ES	0.013	97.4	92.3	97.8	2.0	0.160
CR-CS-IPT	0.013	97.4	92.3	97.8	2.0	0.160
CR-CS-PS	0.013	97.4	92.3	97.8	2.0	0.160
CR-CS-RS	0.013	97.4	92.3	97.8	2.0	0.160
CR-RS-DN	0.035	97.5	92.3	98.0	2.0	0.160
CR-RS-ES	0.035	97.5	92.3	98.0	2.0	0.160
CR-RS-IPT	0.035	97.5	92.3	98.0	2.0	0.160
CR-RS-PS	0.035	97.5	92.3	98.0	2.0	0.160
CR-RS-RS	0.035	97.5	92.3	98.0	2.0	0.160
CR-PS-DN	0.087	97.1	87.1	97.2	2.4	0.151
CR-PS-ES	0.042	97.1	87.1	97.2	2.4	0.151
CR-PS-IPT	0.041	97.1	87.1	97.3	2.4	0.151
CR-PS-PS	0.042	97.1	87.0	97.2	2.5	0.151
CR-PS-RS	0.042	97.1	87.1	97.2	2.4	0.151
Topology						
1 Layer	0.019	95.1	92.5	96.8	2.7	0.159
3 Layer	0.029	98.7	90.6	98.1	1.9	0.158
5 Layer	0.050	98.3	88.5	98.1	1.8	0.154
Alert Rate						
100%	0.041	98.7	96.6	99.3	1.0	0.157
115%	0.024	96.0	84.5	96.0	3.3	0.157
Machines						
250	0.034	97.4	90.6	97.7	2.1	0.157
1000	0.031	97.3	90.5	97.7	2.2	0.157

Appendix II: Cyber Attack Simulator and Results CD

The enclosed CD contains the version of the Cyber Attack Simulator used for this research as well as all the raw data collected from each of the simulation runs. The Cyber Attack Simulator contains two versions, a raw code version and an executable JAR file. The Java Development Kit version 1.5.0 used for this model is also included on the disc. To run the Cyber Attack Executable Jar File, copy the folder "Cyber Attack Executable" to a local computer. Once the "Cyber Attack Executable" folder has been copied to a local machine, open the folder and double click the "CyberAttackSimulator" file to run the simulator.

The six network configurations are included in the save files, and can be accessed through the open network dialog within the Cyber Attack Simulator. Each network file includes the twelve hour attack scenario used for that network. Table II-1 contains a list of the enclosed folders along with a short description.

Table II-1: CD Folder Contents and Description

Folder Name	Description
Cyber Attack Executable	This folder contains all the files necessary to run the cyber attack simulator executable JAR file. Copy this entire folder to a local machine, then double click the "CyberAttackSimulator" file to run the attack simulator.
Cyber Attack Simulator Java Code	This folder contains all the raw java files that a user can use to extend the cyber attack simulator.
Results	This folder contains 12 excel files that hold all the raw data for each simulation run of the experiment. The file names include the network topology, network size, and alert rate of the enclosed data. A number of 175,000 in the file name indicates an alert rate of 100% while a 200,000 indicates an alert rate of 115%. Each file contains four worksheets, each worksheet's name contains the random number seed used in the simulation run to create the data.
JDK	This folder contains the Java development kit used for the creation of the Cyber Attack Simulator.