12-26-2014

Characterizing and Managing Intrusion Detection System (IDS) Alerts with Multi-Server/Multi-Priority Queuing Theory

Christopher C. Olsen

CHARACTERIZING AND MANAGING INTRUSION DETECTION SYSTEM (IDS) ALERTS WITH MULTI-SERVER/MULTI-PRIORITY QUEUING THEORY

THESIS
DECEMBER 2014

Christopher C. Olsen, Captain, USAF

AFIT-ENV-MS-14-D-24

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENV-MS-14-D-24

CHARACTERIZING AND MANAGING INTRUSION DETECTION SYSTEM (IDS) ALERTS WITH MULTI-SERVER/MULTI-PRIORITY QUEUING THEORY

THESIS

Presented to the Faculty

Department of Systems Engineering and Management

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Systems Engineering

Christopher C. Olsen, BS

Captain, USAF

December 2014

DISTRIBUTION STATEMENT A.
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AFIT-ENV-MS-14-D-24

CHARACTERIZING AND MANAGING INTRUSION DETECTION SYSTEM (IDS) ALERTS WITH MULTI-SERVER/MULTI-PRIORITY QUEUING THEORY

Christopher C. Olsen, BS

Captain, USAF

Approved:

________//SIGNED//______________________ _08 Dec 2014_
John M. Colombi, Ph.D. (Chairman) Date

________//SIGNED//______________________ _08 Dec 2014_
Lt Col Kyle F. Oyama, Ph.D. (Member) Date

________//SIGNED//______________________ _08 Dec 2014_
David R. Jacques, Ph.D. (Member) Date

Abstract

The DoD sets forth an objective to “employ an active cyber defense capability to prevent intrusions onto DoD networks and systems.” Intrusion Detection Systems (IDS) are a critical part of network defense architectures, but their alerts can be difficult to manage. This research applies Queuing Theory to the management of IDS alerts, seeking to answer how analysts and priority schemes effect alert processing performance. To characterize the effect of these two variables on queue wait times, a MATLAB simulation was developed to allow parametric analysis under two scenarios. The first varies the number of analysts and the second varies the number of alert priority levels. Results indicate that two analysts bring about drastic improvements (a 41% decrease) in queue wait times (from 116.1 to 49.8 minutes) compared to a single analyst, due to the reduced potential for bottlenecks, with diminishing returns thereafter. In the second scenario, it was found that three priority levels are sufficient to realize the benefits of prioritization, and that a five level priority scheme did not result in shorter queue wait times for Priority 1 alerts. Queuing models offer an effective approach to make IDS resource decisions in keeping with DoD goals for Active Cyber Defense.

To the men and women who serve in silence.

Acknowledgments

There are a variety of faculty and staff at AFIT who I should thank for offering me guidance and encouragement all along the way. The biggest thanks go undeniably to Dr. John Colombi, who took me in when I was late into my thesis hours and added a significant amount of vector to my all-thrust approach. I'd also like to thank Dr. Kenneth Hopkinson from the Department of Electrical and Computer Engineering for helping me in my first, albeit overly ambitious, attempt at a thesis topic. Lt Col Kyle Oyama, as my final professor before beginning my thesis hours, provided a lot of advice and insight into how to select and begin writing a thesis topic. Many thanks to Capt Christina Rusnock for helping to kick start my thesis work, by showing me how to navigate the web of research databases in an organized way. I'd also like to extend much appreciation to Ms. Hallie Owsiany, the distance learning coordinator, for being so awesome the last few years at everything from registration to answering my oddball questions, many of which I'm sure were firmly outside her job description.

I'd be remiss not to thank my parents and grandparents for starting me out in life with the right attitude about and aspirations for higher education. Good grades and advanced degrees were never a discussion of "if" but always of "when" and "why", and for that I can never thank them enough.

And of course, I have to thank my wife (a soon-to-be M.D.!) for letting me work through weekend after weekend when she (and let's be honest, me too) would have rather been doing something a little more…fun.

Captain Christopher Olsen

Table of Contents

Page

List of Figures

Page

List of Tables

Page

AFIT-ENV-MS-14-D-24

CHARACTERIZING AND MANAGING INTRUSION DETECTION SYSTEM ALERTS WITH QUEUING THEORY

I. Introduction

Background

Active Cyber Defense

In the Department of Defense Strategy for Operating in Cyberspace (DSOC), signed in July 2011 by Secretary Robert Gates, the DoD sets forth five strategic initiatives aimed at bringing U.S. cyber defense capabilities to a level that reflects the nation's reliance on cyberspace. Strategic Initiative 2 states that the, "DoD will employ new defense operating concepts to protect DoD networks and systems." Two objectives under this initiative are to "employ an active cyber defense capability to prevent intrusions onto DoD networks and systems" and to develop "new defense operating concepts and computing architectures" (Dept of Defense Chief Information Officer, 2010).

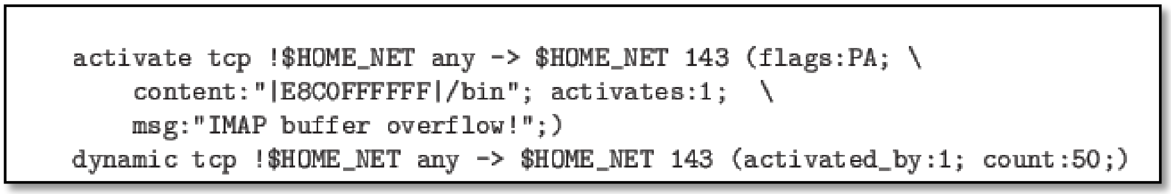
The DSOC describes active cyber defense, or ACD, as a "synchronized, real-time capability to discover, detect, analyze, and mitigate threats and vulnerabilities". The key idea behind ACD is that threat mitigation happens in cyber-relevant time. Cyber-relevant is an intentionally ambiguous descriptor whose value varies depending on the battlespace being discussed. It could be on the order of nanoseconds in the case of a CPU or seconds in the case of end-to-end SATCOM links.

The DSOC also refers to the development of new defense operating concepts in cyberspace. While the context of this statement refers to exploration of mobile media and cloud computing technology, this thesis will characterize the queuing of alerts generated by network intrusion detection systems and use this data to make recommendations about network operation center response options.

Intrusion Detection Systems

Intrusion detection systems (IDS) cover a wide variety of capabilities and implementations. An IDS can be used to detect any kind of undesirable traffic on a network. An IDS provides the same kind of intrusion detection for a computer network as a burglar alarm provides for a house. Traditionally, IDS come in two flavors: network-based IDS (NIDS) and host-based IDS (HIDS). NIDS are placed at a network gateway or along choke points in the network topology to sift through network traffic at large and identify traffic that is malicious, against network policy, or has other undesirable characteristics. HIDS are located at specific network nodes to provide specialized traffic monitoring (Blackwell, 2004). When located at a PC, a HIDS might track central processing unit (CPU) activity or random access memory (RAM) consumption to identify anomalous behavior. A HIDS could also be located at a node providing a specific network service, such as a file transfer protocol (FTP) server or an e-mail server. When employed together, NIDS and HIDS can provide a powerful capability to detect undesirable activity on a computer network.

In simple terms, an IDS works by examining packets as they come across the network and comparing them to a bank of signatures stored within the device. A signature is a specially formatted block of text that tells the IDS what to look for in each packet it inspects. If a packet matches the criteria contained within the signature, the device performs a preprogrammed action against the packet, such as dropping the packet or flagging it for tracking purposes. The IDS usually generates an alert in a human-readable format for the administrator. The management of these alerts is the major topic of this thesis.



```
activate tcp !$HOME_NET any -> $HOME_NET 143 (flags:PA; \
    content:"|E8C0FFFFFF|/bin"; activates:1;  \
    msg:"IMAP buffer overflow!";)
dynamic tcp !$HOME_NET any -> $HOME_NET 143 (activated_by:1; count:50;)
```

Figure 1. An example IDS rule or "signature". Each word sets the value for a configurable parameter, determined by the rule format. This particular format is from the popular open source IDS called SNORT®. (Roesch & Green, 2014)

The set of signatures within an IDS is called its rule-set. While there are numerous open and proprietary sources for rule-sets that will provide protection from an array of common and uncommon threats, the adoption of a rule-set must be done with care. Each network administrator must carefully craft a rule-set that is specialized for their network. Unlike antivirus definitions where a growing database is of little concern due to the discrete and slow arrival times of new files, IDS rule-sets can quickly become so large that they are no longer functional. The first issue is one of processing speed and bandwidth. As each packet enters the IDS it must be compared to every signature in the rule-set in real-time. For example, an in-line IDS residing on a 10 Gbps network could receive anywhere from 800,000 to 15,000,000 packets per second at maximum bandwidth consumption (Schudel). Therefore, the presence of unnecessary rules can quickly overwhelm the IDS processors and create a network logjam. The second issue is that rule-sets that are not focused enough will generate too many alerts and/or false positives. It is the goal of the network administrator to craft a rule-set that permits the free flow of legitimate network traffic and minimizes false positives, while still correctly identifying all malicious or undesirable traffic.

Problem Statement

The numerous types and volume of alerts generated by network IDS drive the need for a disciplined approach in characterizing the flow of alerts to the network administrators dashboard, so that they can be effectively triaged, analyzed, and used to implement defensive actions to keep the network safe and operational.

Research Focus

The focus of this research is to characterize the flow of alerts generated by any number of IDS devices within a network. The specific device that an alert stems from is irrelevant from the analyst perspective. All the data needed to properly respond to the alert is contained within the alert message itself. Therefore, all alerts generated across all IDS residing on the network can flow into a single queue, from which network analysts pull alerts. This is a different approach to today's common practice, which is for alerts to be dumped periodically into a log file, which may not be examined by an administrator or analyst for hours or days.

Understanding the nature of alert flows allows network administrators to make decisions about how many analysts to employ within a network operations center and how alerts should be prioritized in order to ensure alerts are analyzed while they are still relevant to the defense of the network.

Investigative Questions

The key variables associated with alert queuing are the number of analysts servicing alerts, how often alerts arrive into the queue, how long each analyst takes to service an alert, and how many priority levels will be assigned to alerts when configuring the IDS rule-set.

The core investigative question addressed in this thesis is, given varying amounts of traffic density, how does the queuing system performance respond to varying numbers of analysts and priority levels?

Methodology Overview

The methodology used in this thesis is a quantitative stochastic simulation and sensitivity analysis using a validated MATLAB model based in Queuing Theory. This simulation will be used to characterize system performance and generate data for making decisions about network operations and resourcing.

Hypothesis

The addition of more analysts, regardless of the other system variables, will most certainly result in reduced waiting time for alerts. Obviously, however, unlimited resourcing is not practical. So the real question will be, are the system performance gains from the addition of analysts worth the additional resource investment?

When it comes to the number of priority levels, it is expected that giving all alerts equal priority will not lead to ideal system performance, as there are some rare signatures that may trigger catastrophic events. These should always be dealt with first, which is not feasible in a no-priority system. At the other extreme, too many priority levels may add unnecessary complexity with little to gain in terms of performance.

Assumptions/Limitations

As with all models, it is never possible to completely replicate real world conditions. Therefore, care must be taken to try and distill from the infinite pool of variables only those factors that are most important to the phenomenon at hand. More detailed assumptions relating to the specific queuing theory model of this thesis are listed in Chapter III. Predominately, the conclusions are limited by a lack of empirical data regarding arrival rate and service rate distributions for IDS alerts.

Implications

As the DoD transitions toward a posture of Active Cyber Defense, the incorporation of more active network sensors that deliver real-time status updates and alerts to network operators is likely to grow. As these devices become more prolific across the network, the burgeoning data flow could easily overload administrators.

This work attempts to address part of that problem by characterizing the flow of alerts from IDS devices and managing the associated variables such that network operators can take timely action in defense of the DoD and Air Force network.

Organization of Thesis

The remainder of this paper is organized as follows. Chapter 2, Literature Review, discusses common methods of analyzing IDS alerts today, lays out the basic concepts and terminology of Queuing Theory, introduces Little's Law and the analytical equations used in Chapter 3, and discusses some foundations from similar research. Chapter 3, Methodology, describes the research method applied and validates the queuing theory model by comparing sample simulation results to their analytical counterparts. Chapter 4, Analysis and Results, describes two different scenarios and interprets the modeling results in the context of network operations. Chapter 5, Conclusions and Recommendations, summarizes the research results, discusses limitations of the research and potential avenues for future work, and makes recommendations about applying the work to Department of Defense active cyber defense objectives.

II. Literature Review

Chapter Overview

The purpose of this chapter is to review current handling methods and research for network IDS, foundational theory and common practices that will be used to support the investigative questions and methodology in this thesis, and to discuss the useful findings of similar research efforts that this paper will reference or incorporate.

Current Techniques for IDS Alert Management

While IDS are widely used in a variety of networks, there are still common problems administrators face in deploying them effectively. The major problems include:

- **Alert Flooding –** unusually high traffic volume or poorly written rule-sets (which generate excessive false positives) can lead to extremely high volumes of IDS alerts. For large networks, hundreds of thousands of alerts per day can fill up administrator logs (Meyer). Additionally, known network attack vectors exist that allows malicious actors to carry out what is effectively a denial of service attack against IDS systems by flooding logs with useless alerts (Tedesco & Aickelin)
- **Limited Scope of Information –** IDS generate alerts from packet-level data. Therefore, they may not have enough information to report on complex protocol communications or shed light on malicious network activity that spans protocols or networking devices (Blackwell, 2004)
- **False Negatives** – even when signatures are written with due diligence, it is impossible for a network administrator to foresee every possible attack vector or weakness in a network, especially those that could stem from well-resourced or creative adversaries. This can create gaps of "false negatives" in the IDS, where malicious traffic passes through undetected because no signature exists to compare it against. (Meyer, 2008)
- **Problems with Anomaly-Type Signatures** - anomaly signatures are useful because they do not survey individual packets, but instead look at broad trends in network activity across a variety of ports and protocols to determine if something is occurring that is out of the ordinary. However, these signatures depend on having pristine real-world network data from which to create a normalized base to compare against. The problem with using real-world data is one can never be

certain that the data being used to establish the base is not itself contaminated with attacks.

The bulk of research efforts pertaining to IDS and alert management have focused on addressing the problems listed above, improving the ability of IDS to correctly detect malicious traffic, and to reduce the number of alerts produced by the devices to a manageable volume. A sampling of the techniques developed to address these issues is shown below:

- **Alert Correlation and Abstraction –** alert correlation and abstraction is succinctly defined as a "multi-step process that receives alerts from one or more intrusion detection systems as input and produces a high-level description of the malicious activity on the network" (Aziz, 2006). In other words, alerts are compared and lumped into related groups, then compiled into a more human-friendly format for presentation to the user. (Valeur, Vigna, Kruegel, & Kemmerer, 2004)
- **Parsing Programs –** some research has implemented the use of scripts that search the fields of an archive of alerts to delete duplicates and/or alerts with field values that are likely to be false positives (Meyer, 2008). The result is a list of alerts that are more likely to contain alerts that contain information about truly threatening traffic.
- **Alert Correlation with Known Network Vulnerabilities –** investigations have been made into cross-referencing IDS alerts with a database of known network vulnerabilities, to increase the confidence that alerts presented to administrators contain relevant information. (Raulerson, Hopkinson, & Laviers, 2014) (Morin, Me, Debar, & Ducasse)

Once the volume of alerts have been reduced by techniques like those described above, they are usually dumped into a log file (Meyer, 2008). The administrator or network analysts can then go back and review the log file. This approach has the advantage of allowing an analyst with a trained eye to survey a large number of historical alerts in search of unusual patterns. However, the main problem is that this analysis is not conducted in real-time, and does not meet the intent of an "active" cyber defense.

While useful for reviewing network activity over a large chunk of operating time (an entire day or weeks' worth of activity), it is not appropriate for protecting the network in real-time, especially while an attack is underway or during a time of heightened defense posture. Since the status of a network can change very quickly, it may be too late if an analyst receives an alert even more than a few minutes after it is generated. For these times, what is needed is a disciplined way of triaging alerts and presenting them to network analysts so they can be assessed while they are still relevant to the defense of the network. In pursuit of a method to do just that, Queuing Theory is discussed in the following section as a potential solution.

Queuing Theory Primer

Practical Uses

Queuing Theory is the mathematical study of waiting in lines. The primary goals of queuing theory models are to determine the expected length of a line, or queue, and the average amount of time a customer can expect to spend in the system. The term "system" refers to the collective sum of those objects waiting in the queue plus those currently being served. To give a commonplace example, consider the customer experience at a supermarket checkout counter. The customers enter the store, select items for purchase, and then proceed to the checkout area. Upon arrival, the customers selects an available register and stands in line. Each customer can expect to wait in line for some average amount of time before reaching the cashier. Once at the cashier, the customer will again wait some amount of time while items are scanned and payment is taken. When payment is received, the customer exits the store with their goods and the process is complete.

Other applications for queuing theory include:

- Bank teller service lines (Cogdill & Monticino, 2007)
- Telecommunications traffic (Rasch, 1963)
- Traffic intersections
- Telephone customer service centers
- CPU task assignment
- Amusement park rides

Clearly, queuing theory can be a valuable tool for reducing wait times and increasing throughput in a variety of systems. The implications for reduced waste, increased profit, customer satisfaction, and general system improvement continue to drive research into increasingly complex areas of queuing theory and queuing networks.

In the context of the IDS alert management problem, queuing theory offers an excellent framework for building an executable model to demonstrate the generation, flow, and handling of IDS alerts within a network.

Queuing System Characteristics

A queuing system is defined by six independent, or characterizing, variables:

- **A source or population –** consists of all potential objects (customers, tasks, etc.)
- **An arrival process or distribution –** how frequently objects enter the queue
- **A queue or queues –** where objects wait to be assigned to a server
- **One or more servers –** the person or machine that services objects
- **A service time distribution –** how long a server takes to do its task
- **A service discipline –** an algorithm dictating how objects are pulled from the queue

The relations of these variables are shown in Figure 2.

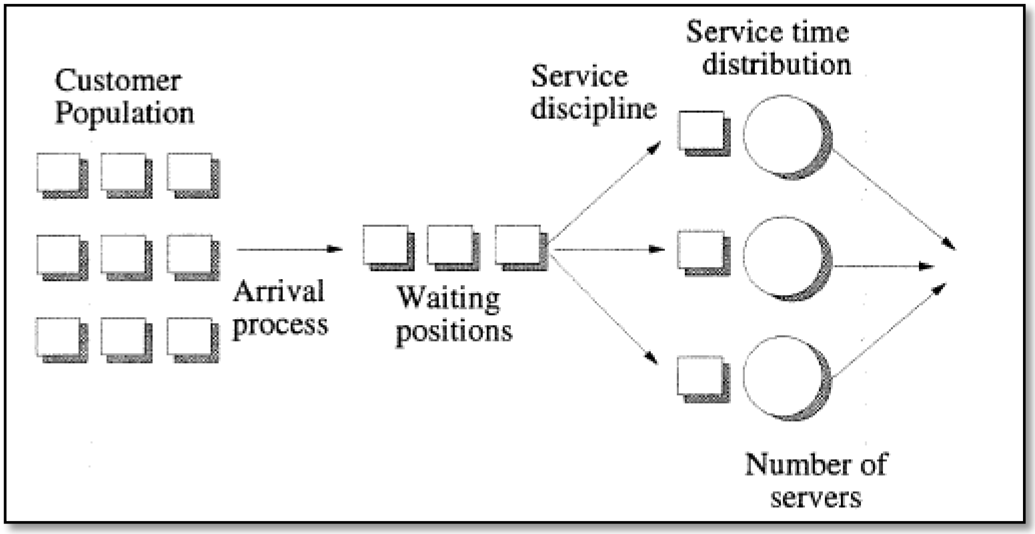


Figure 2. A basic queuing theory diagram calling out the six defining characteristics of a system (Reed, 1995).

The popular Kendall Notation is often used to quickly describe the characteristics of a given queuing system. A generalized Kendall Notation is of the form,

A/S/m/B/K/SD

where,

A is the arrival distribution,

S is the service time distribution,

m is the number of servers,

B is the number of "buffer" spaces or the capacity of the queuing area,

K is the population size, and

SD is the service discipline in use.

Many times, only the first three positions of Kendall Notation are used. In these cases, it can be assumed the buffer capacity and population are infinite and the service discipline is First Come, First Served (FCFS) or First In, First Out (FIFO). A common simple model is M/M/1, where M denotes memoryless (i.e. Markovian) distributions, such as exponential, for inter-arrival and service times, and the system has 1 server.

There are numerous types of distributions, queue/server setups, and service disciplines that can be used to define a queuing system. The variables of particular interest in this paper are the arrival distribution, service distribution, number of servers, and the service discipline.

Once a system is characterized, the values of four dependent variables can be determined:

- L = Total number of objects in the system
  - L_q = Number of objects waiting in the queue(s)
  - L_s = Number of objects receiving service
- W = Amount of time an object spends in the system
  - W_q = Time spent waiting in the queue
  - W_s = Time spent receiving service

Data concerning jobs in the system can provide information about system capacity and whether the system is overcrowded or underutilized. Similarly, data regarding wait times and service times can be useful metrics for determining system efficiency and identifying process bottlenecks.

It is possible, for simple queuing systems, to analytically develop a set of governing equations which determine the values for the dependent variables listed above. These equations will be introduced in the next section and demonstrated in Chapter III to prove the accuracy and calibration of the alert queuing model under simple conditions. For the alert management problem explored in Chapter IV, however, complexity makes the development of analytical solutions infeasible. Therefore, a model provides a mechanism to gather empirical data and provide insight beyond the reach of analytical closed-form solutions.

Little’s Law and Queuing Theory Analytical Equations

A foundational law for solving queuing problems analytically is Little’s Law, first proved by John D.C. Little in 1961. It states,

> The time average number of customers in a queuing system, L, is equal to the rate at which customers arrive and enter the system, λ, times the average sojourn time of a customer, W (Sigman)

Or simply,

$$L = \lambda W$$

Using Little’s law as a foundation, governing equations can be derived for many simple queuing systems. Table 1 shows the governing equations for M/M/1, M/M/c, and M/M/1 with k-Priority queuing systems.

For an M/M/c system where multiple servers are utilized, another variable is introduced, $\rho_{0,0}$, which is the probability that at any given time there are zero alerts waiting in the queue and zero busy analysts. This value is used in computing the average length of the queue, Lq, from which the other variables are determined easily using Little’s Law.

When a First Come, First Served service discipline with priority is introduced (M/M/1 w/k-Priority in Table 1) it is necessary to calculate $\bar{R}$, the mean residual service time in the system. Alerts of lesser priority will have to wait for all alerts of higher priority to clear the system before being serviced. The mean residual service time is the sum of this extra waiting time for lower classes introduced by the prioritized service discipline. Once $\bar{R}$ is calculated, the simplest approach is the calculate how long alerts of each priority will wait in the queue, W_q^K. Determining the values of the other variables is then straightforward.

It was once thought that analytically determining the dependent variables for an M/M/c w/k-Priority system was not feasible, because mean residual service times would become too difficult to determine if each k priority level had a different mean service time (Virtamo). However, a 2005 paper by Harchol-Balter et al introduced Recursive Dimensionality Reduction (RDR) as an analytical approach to analyze multi-server queuing systems with multiple priority classes (Harchol-Balter, Osogami, Scheller-Wolf, & Wierman, 2005) RDR was shown to be accurate to modeling within 2%. However, the effectiveness of RDR was reduced with increasing numbers of servers and priority levels. Thus, the value of a dynamic queuing model that can handle varying service time parameters for differing numbers of servers and priority levels becomes apparent, since it is possible to yield results that can be too complex for analytical investigation.

Table 1. Governing equations for M/M/1, M/M/c, and M/M/1 with k-Priority queuing

Variable	M/M/1	M/M/c	M/M/1 w/k-Priority
L	$= \lambda W$	$= Lq + \frac{\lambda}{\mu}$	$L = L_q + \frac{\lambda_1}{\mu} + \cdots + \frac{\lambda_k}{\mu}$ $L^k = L_q^k + \frac{\lambda_k}{\mu}$
Lq	$= \lambda Wq$	$= (\rho_{0,0}) \left[\frac{\left(\frac{\lambda}{\mu}\right)^{c+1}}{(c-1)!\left(C - \frac{\lambda}{\mu}\right)^2} \right]$	$Lq = L_q^1 + \cdots + L_q^k$ $Lq^k = \lambda_k W_q^k$
W	$= \frac{1}{\mu - \lambda}$	$= \frac{L}{\lambda}$	$W = \frac{\lambda_1 W^1 + \cdots + \lambda_k W^k}{\lambda_1 + \cdots + \lambda_k}$ $W_K = W_q^k + \frac{1}{\mu}$
Wq	$= W - \frac{1}{\mu}$	$= \frac{Lq}{\lambda}$	$W_q = \frac{\lambda_1 W_q^1 + \cdots + \lambda_k W_q^k}{\lambda_1 + \cdots + \lambda_k}$ $W_q^K = \frac{\bar{R}}{(1 - \rho_1 - \cdots - \rho_{k-1})(1 - \rho_1 - \cdots - \rho_k)}$
$\rho_{0,0}$	---	$= \frac{1}{\left(\frac{\lambda}{\mu}\right)^c \left(\frac{1}{c!}\right) \left[\frac{1}{1-\rho}\right] + \sum_{r=0}^{r=c-1} \left(\frac{\lambda}{\mu}\right)^r \left(\frac{1}{r!}\right)}$	---
$\bar{R}$	---	---	$= \frac{1}{2} \sum_{k=1}^{k} \lambda_k \overline{S_k^2}$
$\overline{S_k^2}$	---	---	$= \frac{n!}{\mu^2}$

Foundations from Similar Research

Single Queue, Multiple Servers

The queuing model used in this research relies on the premise that a single queue feeding multiple servers operates more efficiently (i.e. with reduced sojourn time through the system and thus less wait time in the queue) than multiple servers each fed by their own queue.

Figure 3, adapted from a publicly available lecture from University of Illinois (Reed, 1995), compares the mean response time, or system transit time *W*, of multiple servers each fed by their own queue (m M/M/1 queues shown in red) versus that of multiple servers fed by a single queue (1 M/M/m, or 1 M/M/c, queue shown in green). For all server quantities greater than one, a single M/M/c queue shows improved performance over multiple M/M/1 queues. At ten servers, the M/M/c queue processes objects through the queue approximately 70% faster than multiple M/M/1 queues.

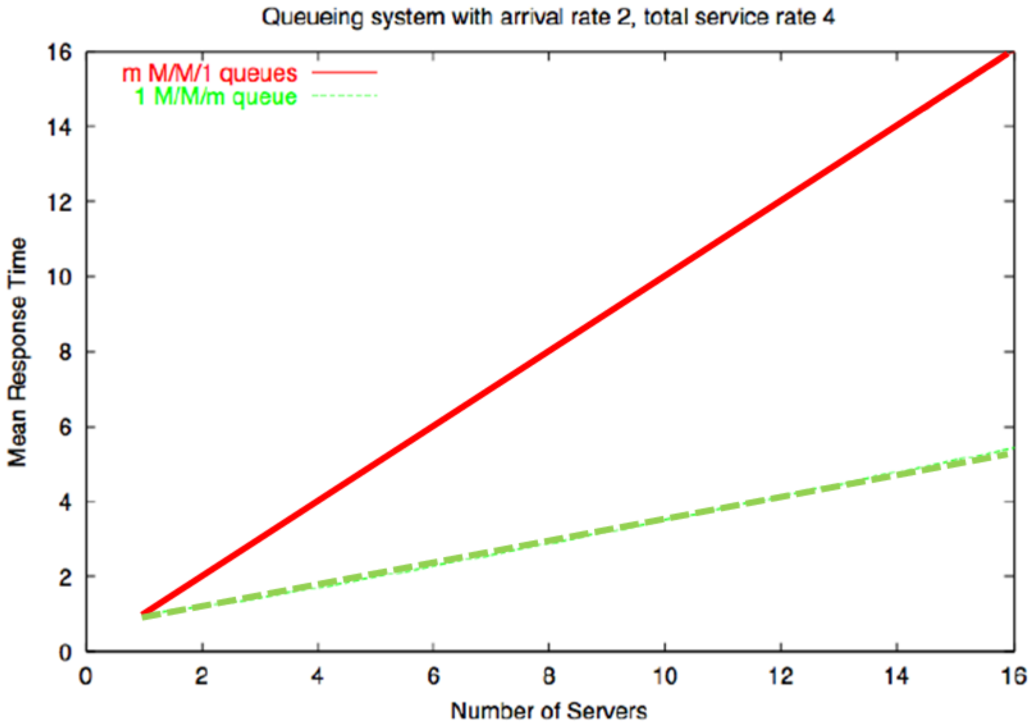


Figure 3. Comparison of the mean response time, or system transit time W, for multiple servers fed by multiple queues (shown in red) versus multiple servers fed by a single queue (shown in green) (Reed, 1995).

Summary

This chapter presented the current techniques for IDS alert management and the research being conducted to improve these systems. It also discussed the basics of Queuing Theory, which is a simple but powerful tool that can be applied to a variety of situations where objects or tasks must wait to be serviced. The analytical mathematics can become difficult with increasing model complexity, but can be calculated for simple systems to a sufficient level for cross-checking results from computer simulations. The queuing model written for this research relies on the principal that a single queue feeding multiple servers is more efficient than multiple servers, each with its own queue. The next section will discuss the methodology for this research and demonstrate the validity of the MATLAB model.

III. Methodology

Chapter Overview

The purpose of this chapter is to describe the research approach and modeling procedures. First a brief overview of the methodology will be conducted, followed by a discussion of the relevant variables. Next, the custom queue model used will be described and then data collected from the model will be compared against theoretical values to prove model accuracy and calibration. Finally, significant underlying assumptions to the experimental process will be discussed.

Overview of Research Methodology

The methodology employed was a computer based simulation model written by the author within the MATLAB computing environment (version R2013a). Since the primary objective of the research was to characterize Alert queuing for Intrusion Detection Systems, the model was founded in Queuing Theory principles as presented in the Queuing Theory primer in Chapter II. This led to the creation of a flexible model which was used to generate data for parametric analysis in determining the sensitivity of the system to changes in the relevant Queuing Theory variables.

Description of Dependent and Independent Variables

In order to provide flexibility, the model allows for several user-defined variables.

The duration of the simulation is determined by the number of arrivals of the lowest level priority alert, which the user defines. The model will continue to run until that number of arrivals is reached.

Users must also define other key Queuing Theory parameters including the number servers (Analysts in this case), the number of Alert priority levels, the inter-arrival rate for each priority level (1/λ), and the service time parameter for each priority level (1/μ). In the version used in this paper, inter-arrival and service time distributions are limited to the exponential form.

Experimental Design

This research uses a “validate and extend” approach to ensure the modeling results on which analysis is conducted are reliable. The first step was to run the simulation under conditions that were easily calculated with the analytical closed-form equations introduced in Chapter 2. The computer-based modeling results were then compared to these analytical equations to ensure the program was producing accurate results. The results of these calibration runs are presented later in this chapter.

Once the model was validated, the parameter values were extended beyond the reach of analytical equations, to generate empirical data for parametric analysis. The analysis was used to characterize the effect of dependent variables on overall system performance and finally to draw practical conclusions about applications to network operations.

Experimental Tasks

1) Design and write a MATLAB model.

2) Develop the set of analytical equations.

3) Validate the MATLAB model.

4) Make parameter assumptions for Scenario 1.

5) Run Scenario 1.

6) Analyze the results from Scenario 1

7) Make parameter assumptions for Scenario 2.

8) Run Scenario 2.

9) Analyze the results from Scenario 2.

Description of Analysis

Analysis was divided into two phases. The first phase compared model output under simple conditions to results predicted by analytical equations. This demonstrated the model was accurate and calibrated. This step was necessary to provide assurance that future data output based on more complex conditions, which cannot be checked analytically, was reliable.

The second phase runs the model under more complex, realistic conditions. The model is run with careful variation in parameters to generate 3D surface plots that demonstrate the sensitivity of the system. The second phase of analysis was divided into two Scenarios that will be discussed in Chapter 4.

Construction of Model

The complete MATLAB code for the model used in this research is included in the Appendix, but a brief description of how the model works is provided. The model uses object-oriented programming techniques to generate a functioning queue and set of servers (or analysts). A main script orchestrates alert arrivals, queue management, analyst tasking, and metric/statistical logging. The main script allows the user to define the total number of alert arrivals (which equates to the length of the simulation), the number of analysts, the number of priority levels, and to define an arrival rate and service rate for each priority level.

An alert class is used to construct alert objects, each of which has 4 properties associated with it: the alert priority level (`priority)`, the time it arrived into the queue (`time_arrived`), the time it was polled (or pulled) from the queue (`time_polled`), and the time it reached service completion and exited the system (`time_finished`).

A queue class is used to construct the queue object(s). Queue objects have properties for `type`, which is always of type alert objects in this research, and individual `elements`, which are the queue locations that each alert is placed into.

Finally, there is an analyst class, which creates analyst objects that have a Boolean property of being available or busy (`available`) and an `alert` property, which is where they store the current alert they are servicing.

Figure 4 is a class diagram to show how the various classes interact with the main script to produce a complete queuing theory model.

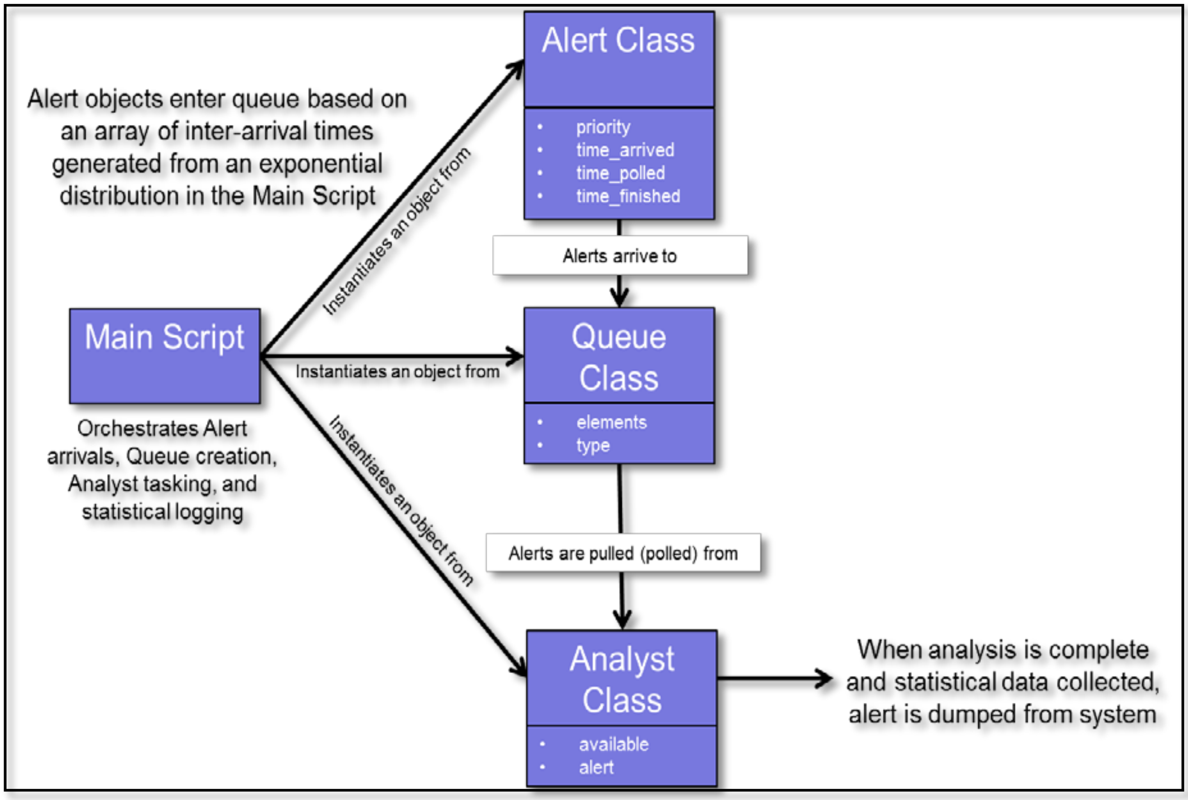

Alert Class
priority
time_arrived
time_polled
time_finished
Alert objects enter queue based on an array of inter-arrival times generated from an exponential distribution in the Main Script
Instantiates an object from
Alerts arrive to
Main Script
Instantiates an object from
Queue Class
elements
type
Orchestrates Alert arrivals, Queue creation, Analyst tasking, and statistical logging
Instantiates an object from
Alerts are pulled (polled) from
Analyst Class
available
alert
When analysis is complete and statistical data collected, alert is dumped from system

Figure 4. Class diagram depicting how the Main Script of the model orchestrates between the three classes to create a functioning queuing system.

Validation of Model

For assurance that the model generates reliable and accurate data, model output was compared with Queuing Theory analytical results for a set of simple parameters.

Before proceeding, recall from Chapter I that Kendall notation of the form A/S/m/B/K/SD is the standardized method of describing queuing systems. Many times, only the first three positions of Kendall Notation are used. In these cases, it can be assumed the buffer capacity and population are infinite and the service discipline is First Come, First Served (FCFS). Thus, M/M/1 = M/M/1/∞/∞/FCFS.

M/M/1Analytical Results

Perhaps the simplest Queuing Theory system is the M/M/1, where M represents an exponential distribution. Therefore, in an M/M/1 system Alert inter-arrivals and service times are based on an exponential distribution and there is only one server, or Analyst.

For comparison to model results, assume an arrival rate of λ = 10 alerts per hour (1/λ = 1/10, or an alert arrives on average every 6 minutes) and a service rate of μ=12 alerts per hour (1/μ = 1/12, or one alert is serviced on average every 5 minutes). Therefore, the average time each alert spends in the system (sitting in the queue plus the time it takes to be serviced), *W*, is

$$\boldsymbol{W = \frac{1}{\mu - \lambda} = \frac{1}{12 - 10} = \frac{1}{2}\ or\ .5\ hours\ (30\ mins)}$$

the average time each alert sits in the queue before getting serviced, *Wq*, is

$$\boldsymbol{Wq = W - \frac{1}{\mu} = \frac{1}{2} - \frac{1}{12} = \frac{5}{12}\ or\ .417\ hours}\ \textbf{(25 mins)}$$

the average number of alerts in the system, *L*, is

$$\boldsymbol{L = \lambda W = (10)\left(\frac{1}{2}\right) = \frac{10}{2} = 5\ alerts}$$

and the average number of alerts in the queue, *Lq*, is

$$\boldsymbol{Lq = \lambda Wq = (10)\left(\frac{5}{12}\right) = \frac{50}{12} = 4.167\ alerts}$$

M/M/1 Model Results

The data in Figure 5 shows convergence to theoretical values over a trial run of 100,000 alert arrivals using the same values of λ=10 and μ=12. The red reference lines represent the theoretical values.

It should be noted queue size, Lq, and system size, L, throughout this analysis are based on a sampling of simulation time, anywhere from every 1,000 to 10,000 minutes depending on the complexity and total simulation time of the model. Due to the varying arrival rates and priority levels in use, simulation time could expand to millions of time steps, with a separate array required for each priority level. Continuous evaluation of the average queue and system size for every iteration of simulation time proved to be too computationally intense for the home PC to handle.

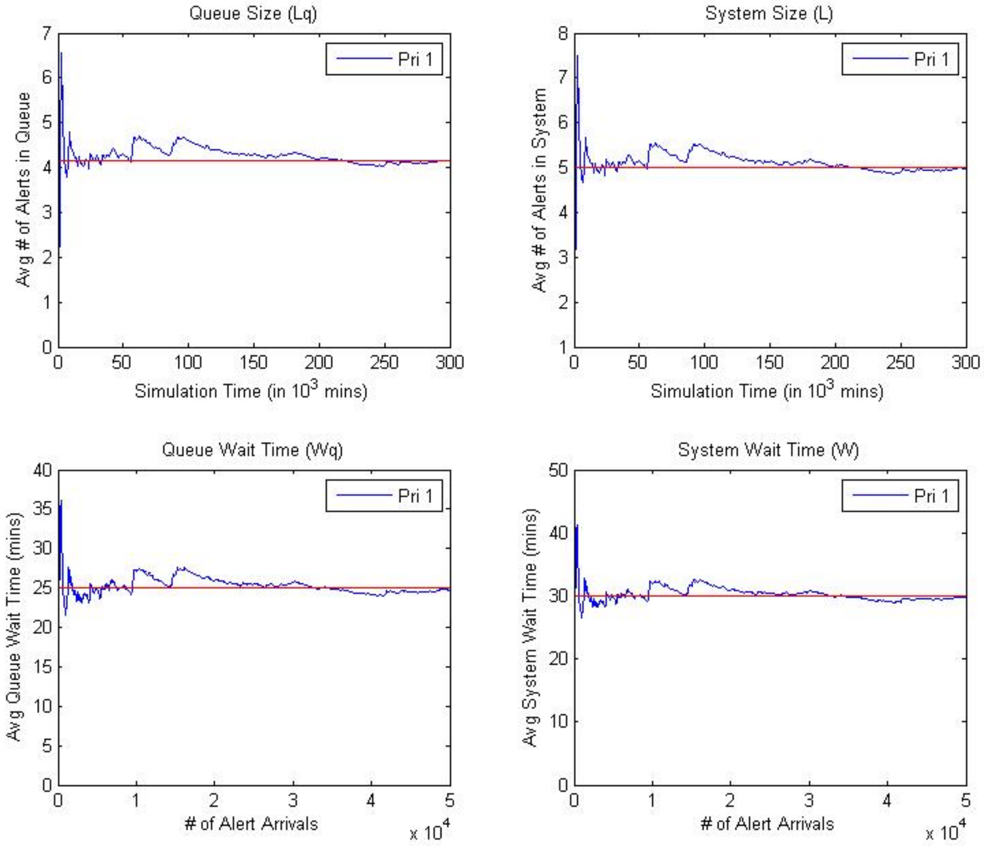


Figure 5. Clockwise, the average Queue Size, System Size, System Wait Time, and Queue Wait Time output by the model for a simple M/M/1 system (λ=10, μ=12).

At steady state, the model converges to the theoretical reference lines pictured. There is, however, a transient period where the average size and wait time values fluctuate drastically. In order to negate the effects of the transient period when calculating model error, only the last 10% of the curve data points are used when calculating root-mean-squared (RMS) error.

The average values of the response variables as predicted by the model are compared to the theoretical values in Table 2. The small RMS errors suggest the model is an excellent predictor of queue behavior for the M/M/1 case.

Table 2. Comparison of the average system dependent variable values output by the model compared to theoretical predictions, for an M/M/1 system with λ=10 and μ=12.

System Characteristic	Model Steady State	Theoretical Value	RMS Error
Lq	4.143	4.167	.0515
L	4.976	5.000	.0521
W	29.70	30.00	0.4617
Wq	24.73	25.00	0.4313

M/M/c Analytical Results

The next step to increase the complexity of the model is to allow any multiple analysts to pull alerts from the queue. Arrival and service distributions remain in the exponential form.

To provide some values for comparison to the model output, assume λ = 3 alerts per hour (1/λ = 1/3, or an alert arrives on average every 20 minutes), μ=6 alerts per hour

(1/μ = 1/6, or an alert is serviced on average every 10 minutes), and c=3 (three Analysts are pulling alerts from the queue).

Therefore, using the M/M/c equations introduced in Chapter 1, ρ for the system is,

$$\rho = \frac{\lambda}{c\mu} = \frac{3}{(3)(6)} = \frac{3}{18} = \frac{1}{6}$$

and the probability that there are zero alerts waiting in the queue and zero busy analysts is,

$$\rho_{0,0} = \frac{1}{\left(\frac{\lambda}{\mu}\right)^{c}\left(\frac{1}{c!}\right)\left[\frac{1}{1-\rho}\right] + \sum_{r=0}^{r=c-1}\left(\frac{\lambda}{\mu}\right)^{r}\left(\frac{1}{r!}\right)} =$$

$$= \frac{1}{\left(\frac{3}{6}\right)^{3}\left(\frac{1}{3!}\right)\left[\frac{1}{1-\frac{1}{6}}\right] + \left(\frac{3}{6}\right)^{0}\left(\frac{1}{0!}\right) + \left(\frac{3}{6}\right)^{1}\left(\frac{1}{1!}\right) + \left(\frac{3}{6}\right)^{2}\left(\frac{1}{2!}\right)} = .60\overline{60}$$

So the predicted values for Lq, L, Wq, and W are

$$\boldsymbol{Lq = (\rho_{0,0})\left[\frac{\left(\frac{\lambda}{\mu}\right)^{c+1}}{(c-1)!\left(c-\frac{\lambda}{\mu}\right)^{2}}\right] = (.60\overline{60})\left[\frac{\left(\frac{3}{6}\right)^{3+1}}{(3-1)!\left(3-\frac{3}{6}\right)^{2}}\right] =.0030\, alerts}$$

$$\boldsymbol{L = Lq + \frac{\lambda}{\mu} =.0030 + \frac{3}{6} =.5030\, alerts}$$

$$\boldsymbol{Wq = \frac{Lq}{\lambda} = \frac{.0030}{3} =.0010\, hours\ or.0606\, mins}$$

$$\boldsymbol{W = \frac{L}{\lambda} = \frac{.5030}{3} =.1677\, hours\ or 10\, mins}$$

M/M/c Model Results

Figure 6 displays data from a run of the model with λ=3, μ=6, and c=3 analysts. The values of the four response variables converge to the analytically predicted values, which are marked by the red reference lines. Due to the relatively low density of the system (ρ=1/6), convergence is not as complete as in the M/M/1 case above. Still, the model output agrees nicely with theoretical values.

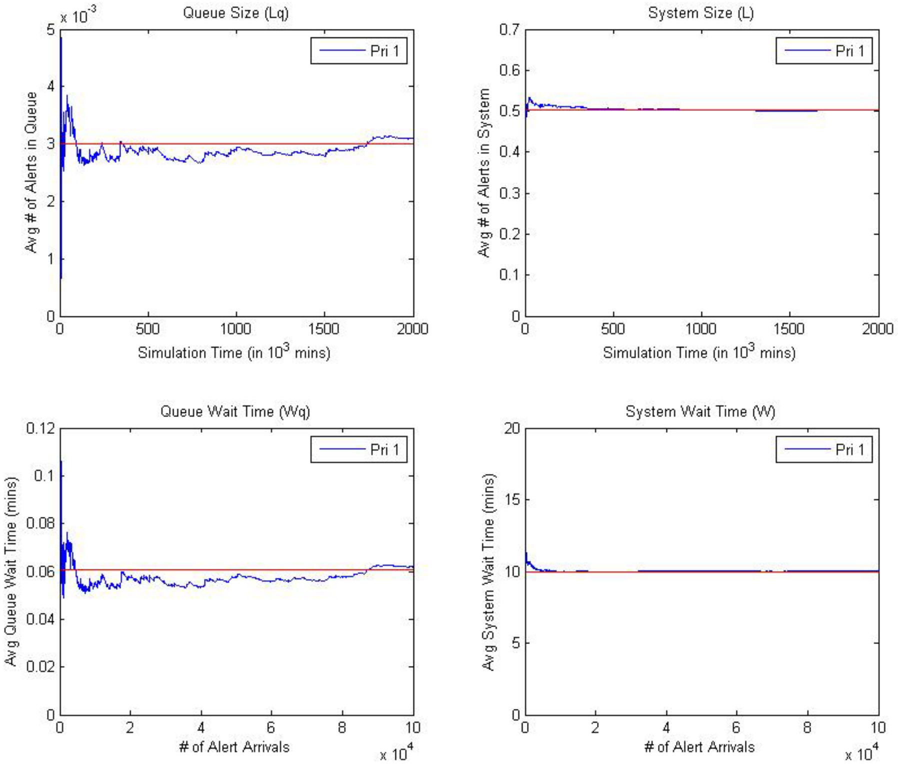


Figure 6. For an M/M/3 system with λ=3 and μ=6, the model agrees with analytical predictions of the four response variables.

Table 3. Comparison of the average system dependent variable values output by the model compared to theoretical predictions, for an M/M/3 system with λ=3 and μ=6.

System Characteristic	Model Steady State	Theoretical Value	RMS Error
Lq	.0031	.0030	0.0001
L	.5031	.5030	0.0006
W	10.05	10.00	0.0526
Wq	.0618	.0606	0.0016

M/M/1 with Priority Analytical Results

Now the model will return to a single analyst, but multiple alert priority levels will be added. To keep the calculations manageable, analytical results will be shown for two (2) priority levels, with $\lambda_2 = 3$, $\lambda_1 = 2$, $\mu = 12$, and $c = 1$ where λ_1 is high priority and λ_2 is low.

From the M/M/1 with Priority equations in Chapter 1, the first thing that must be calculated is the second moment of the service distribution, S_k^2,

$$\overline{S_k^2} = \frac{n!}{\mu^2} = \frac{2!}{12^2} = \frac{1}{72} or. 0139$$

which is plugged into the mean residual service time parameter,

$$\bar{R} = \frac{1}{2}\sum_{k=1}^{k} \lambda_k \overline{S_k^2} = \frac{1}{2}\left[(3)\left(\frac{1}{72}\right) + (2)\left(\frac{1}{72}\right)\right] = .0347$$

The residual service time is used to estimate the average time spent in the queue for each priority level,

$$W_q^1 = \frac{\bar{R}}{1-\rho_2} = \frac{.0347}{1-\frac{2}{12}} = .0417\ hours\ or\ 2.500\ mins$$

$$W_q^2 = \frac{\bar{R}}{(1-\rho_2)(1-\rho_2-\rho_1)} = \frac{.0347}{\left(1-\frac{2}{12}\right)\left(1-\frac{2}{12}-\frac{3}{12}\right)}$$

$$= .0714\ hours\ or\ 4.286\ mins$$

then, the average amount of time any alert spends in the queue (the system-wide queue wait time) is calculated using a weighted approach,

$$\boldsymbol{Wq = \frac{\lambda_2 W_q^2 + \lambda_1 W_q^1}{\lambda_2 + \lambda_1} = \frac{(3)(4.286) + (2)(2.5)}{3+2} = 3.571\ mins}$$

System sojourn time, W, is calculated for each priority level by adding the expected time the alert will wait in the queue plus the average time the alert takes to be serviced,

$$W^1 = W_q^1 + \frac{1}{\mu} = .0417 + \left(\frac{1}{12}\right) = .1250\ hours\ or\ 7.502\ mins$$

$$W^2 = W_q^2 + \frac{1}{\mu} = .0714 + \left(\frac{1}{12}\right) = .1547\ hours\ or\ 9.286\ mins$$

and similarly to Wq,

$$\boldsymbol{W = \frac{\lambda_2 W^2 + \lambda_1 W^1}{\lambda_2 + \lambda_1} = \frac{(3)(9.286) + (2)(7.502)}{3+2} = 8.571\ mins}$$

Finding the queue size for each alert priority level requires only a quick application of Little’s Law,

$$Lq^1 = \lambda_1 W_q^1 = (2).0417) = .0833\ priority\ 1\ alerts$$

$$Lq^2 = \lambda_2 W_q^2 = (3)(.0714) = .2143\ priority\ 2\ alerts$$

And the overall queue size, Lq, regardless of priority is a simple addition,

$$\boldsymbol{Lq = Lq^2 + Lq^1 = .2143 + .0833 = .2976\ alerts}$$

The system size, L, for each alert priority level is simply the number of alerts of that priority in the queue plus the average arrival rate,

$$L^1 = L_q^1 + \frac{\lambda_1}{\mu} = .0833 + \frac{2}{12} = .2500\ alerts$$

$$L^2 = L_q^2 + \frac{\lambda_2}{\mu} = .2143 + \frac{3}{12} = .4643\ alerts$$

Making the overall system size,

$$\boldsymbol{L = L_q + \frac{3}{12} + \frac{2}{12} = .2976 + \frac{3}{12} + \frac{2}{12} = .7143\ alerts}$$

M/M/1 with Priority Model Results

When using multiple priority levels, each level had its own value for Lq, L, Wq, and W but there was also a system value that looked at the characterization of the system as a whole without regard for priority levels. Figure 7 below shows the convergence of the model to the theoretical average values predicted above for each priority level as well as for the system as whole, which is shown in black. Red reference lines on the charts represent the closed-form analytical predictions.

One may notice that the Priority 1 and System curves do not extend the full length of the X axis in the Wq and W plots, whereas the Priority 2 curve does. This is simply a result of the way the model handles alert arrivals of differing priority levels. Since the total simulation length is determined based on a given number of arrivals for the lowest priority (in the case, Priority 2), the number of higher priority arrivals will vary based on its arrival rate. Since higher priority alerts generally arrive less frequently than low ones, there will be less total high priority arrivals, but they will still enter the system throughout the entire simulated time span.

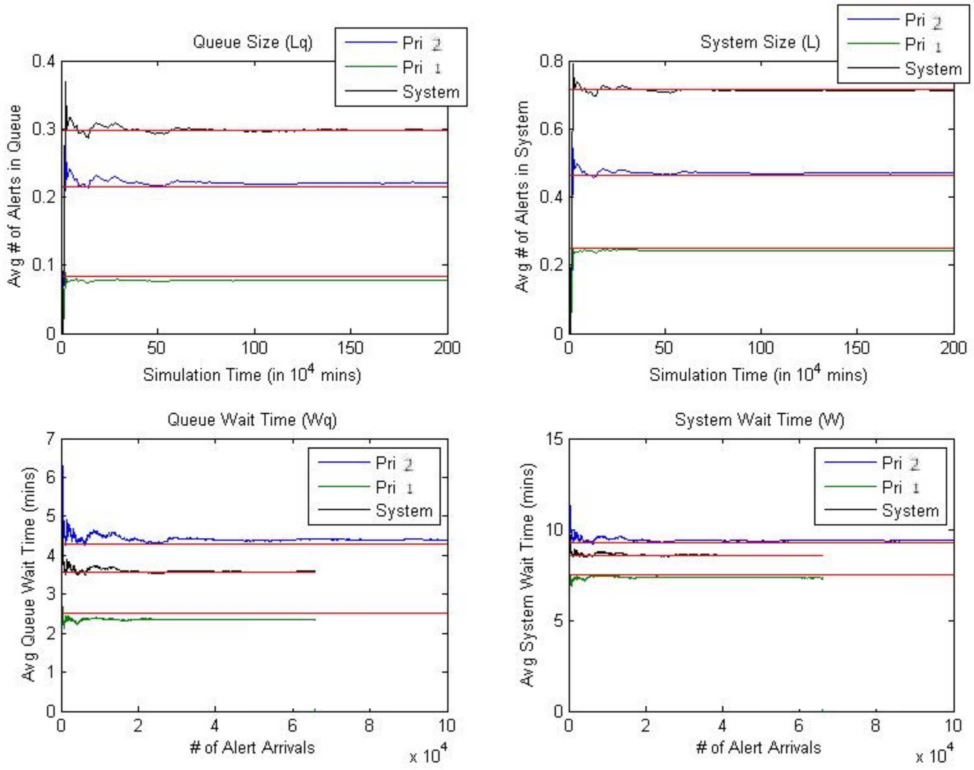


Figure 7. Model output for the four response variables for an M/M/1 with 2 Priority system. Red reference lines represent values predicted by analytical equations.

Table 4 compares the average values determined by the model to the theoretical values. For system-wide variables, error appears to be slightly less than for individual priority levels. As was seen for the M/M/3 system above, the higher error for the individual priority levels is influenced by the lower traffic density.

Table 4. Comparison of the model and theoretical dependent variable average values, both overall and for each priority level, in an M/M/1 with 2 priority level system.

System Characteristic	Model Steady State	Theoretical Value	RMS Error
$\boldsymbol{Lq}$	.2976	.2976	0.0009
L_q^2	.2203	.2143	0.0067
L_q^1	.0774	.0833	0.0059
$\boldsymbol{L}$	.7123	.7143	0.0012
L^2	.4703	.4643	0.0068
L^1	.2420	.2500	0.0078
$\boldsymbol{W}$	8.570	8.571	0.0054
W^2	9.374	9.286	0.1014
W^1	7.325	7.500	0.1732
$\boldsymbol{Wq}$	3.592	3.571	0.0107
W_q^2	4.391	4.286	0.1174
W_q^1	2.341	2.500	0.1578

Assumptions

Many assumptions were made in the design and execution of the MATLAB queuing theory model. Some of the major assumptions were:

- All alerts that reach the queue are valid alerts and not the results of false positive triggers. All network traffic and detection activity that happens “upstream” from the system is not included in the model, which starts at the alert queue and ends after the analyst services the alert.

- The network analyst is well-trained and has already traversed the steepest portion of the learning curve. In other words, the analyst is proficient at servicing alerts and spends only as much time as necessary on each one.

- In multiple server (i.e. multiple analyst) scenarios, each analyst has the same skill proficiency. In the real world, some analysts will have more advanced skills, so service times may vary drastically from analyst to analyst and may not be solely a function of the type of alert.

- There is assumed to be no collaboration between analysts. For example, the findings of one analyst does not affect the work of the others. Even though the alerts in service could have related root causes.

- The model does not allow alerts to be preempted. Alerts of the highest priority must wait at the front of the queue until an analyst finishes working on a lower priority alert. In the real world, a lower priority alert may be set aside when an exceptionally critical one arrives.

Summary

The method used in this research begins with validating the MATLAB model by comparing its results under simple conditions to the results predicted by closed-form analytical equations. Once the model is verified, its parameters are extended to simulate conditions beyond the reach of analytical equations. The results of these extensions and their interpretation are presented in the following chapter.

IV. Analysis and Results

Chapter Overview

This chapter introduces the modeling results for two different trial scenarios. In each scenario, 3D surface plots are generated to determine the sensitivity of the system to three key parameters: traffic density, number of analysts, and number of priority levels. The same set of traffic densities were used in both Scenario 1 and 2, ranging from lighter to heavier loads to see how the system would respond under varying amounts of stress. In Scenario 1 the model was executed across all traffic loads while varying the number of analysts from one (1) to three (3). In Scenario 2, the results of Scenario 1 under the three analyst case were extended using the same set of traffic loads, but with one (1), three (3), and five (5) priority levels.

Scenario 1: Varying the Number of Analysts

For the first scenario, the number of analysts was varied between one (1) , two (2), and (3) analysts across a range of traffic loads from lighter to heavier to determine the effect on average queue size (Lq), average system size (L), average queue wait time (Wq), and average system sojourn time (W). All values for alert arrival rate (λ) and alert service rate (μ) followed an exponential distribution.

Scenario1: Assumptions

For Scenario 1, the number of priority levels was fixed at three (one through three, with priority level one being high and priority three low) to make it easier to identify the sensitivity of the system to varying the number of analysts only, across a range of traffic loads. The values for alert arrival rate (λ) and alert service rate (μ) for each priority level and traffic load are summarized in Table 5.

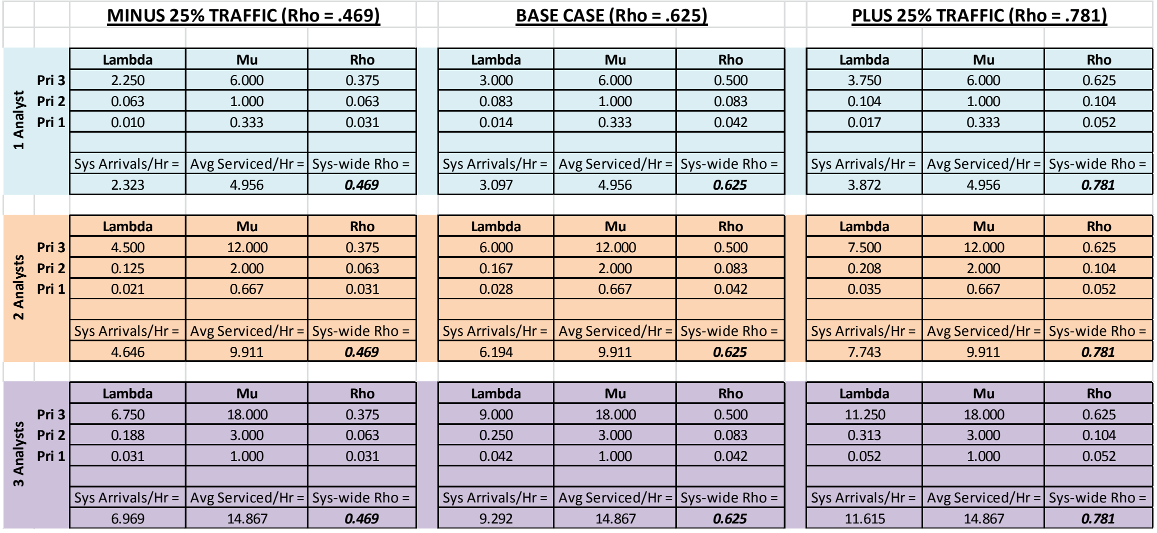
The base case for traffic load was created from assumptions about what would constitute reasonable arrival and service times for various types of alerts. Priority one alerts should be relatively rare events, so it is assumed that a priority one alert will only arrive on average once per 72-hour period, yielding $\lambda=1/72$. When they do occur, however, these events will require significantly more time to service since they pose a major risk to network operations and likely stem from complex or novel traffic phenomena. In this scenario their service time was set to an average of three hours, or $\mu=1/3$.

Priority two events will occur more frequently than priority one events, but less than the common priority three events, so they were set to arrive about once per 12-hour period, or $\lambda=1/12$. These events are assumed to take less time to service since they pose a lower risk to network operations and are likely simpler to analyze, so their service time was set at $\mu=1$, or 1 alert per hour.

Finally, the common priority three alerts are generally considered low priority because they represent minimal risk to network operations. Since many can be dismissed without careful consideration, their average service time is quite quick, and has been assumed here as 10 minutes, or $\lambda=6$ alerts per hour.

To reduce and increase the traffic load, $\rho=\lambda/\mu$, the base values described above for alert arrivals, λ, were decreased and increased by 25%, respectively, across all priority levels. As analysts were added to the system, the corresponding service capacity automatically increased, so the lambda values were doubled or tripled as necessary to ensure the same stress was applied regardless of the number of analysts employed.

Table 5. Summary of the model parameters used to generate each of the nine cases in Scenario 1.



		MINUS 25% TRAFFIC (Rho = .469)			BASE CASE (Rho = .625)			PLUS 25% TRAFFIC (Rho = .781)		
1 Analyst		Lambda	Mu	Rho	Lambda	Mu	Rho	Lambda	Mu	Rho
	Pri 3	2.250	6.000	0.375	3.000	6.000	0.500	3.750	6.000	0.625
	Pri 2	0.063	1.000	0.063	0.083	1.000	0.083	0.104	1.000	0.104
	Pri 1	0.010	0.333	0.031	0.014	0.333	0.042	0.017	0.333	0.052
		Sys Arrivals/Hr =	Avg Serviced/Hr =	Sys-wide Rho =	Sys Arrivals/Hr =	Avg Serviced/Hr =	Sys-wide Rho =	Sys Arrivals/Hr =	Avg Serviced/Hr =	Sys-wide Rho =
		2.323	4.956	***0.469***	3.097	4.956	***0.625***	3.872	4.956	***0.781***
2 Analysts		Lambda	Mu	Rho	Lambda	Mu	Rho	Lambda	Mu	Rho
	Pri 3	4.500	12.000	0.375	6.000	12.000	0.500	7.500	12.000	0.625
	Pri 2	0.125	2.000	0.063	0.167	2.000	0.083	0.208	2.000	0.104
	Pri 1	0.021	0.667	0.031	0.028	0.667	0.042	0.035	0.667	0.052
		Sys Arrivals/Hr =	Avg Serviced/Hr =	Sys-wide Rho =	Sys Arrivals/Hr =	Avg Serviced/Hr =	Sys-wide Rho =	Sys Arrivals/Hr =	Avg Serviced/Hr =	Sys-wide Rho =
		4.646	9.911	***0.469***	6.194	9.911	***0.625***	7.743	9.911	***0.781***
3 Analysts		Lambda	Mu	Rho	Lambda	Mu	Rho	Lambda	Mu	Rho
	Pri 3	6.750	18.000	0.375	9.000	18.000	0.500	11.250	18.000	0.625
	Pri 2	0.188	3.000	0.063	0.250	3.000	0.083	0.313	3.000	0.104
	Pri 1	0.031	1.000	0.031	0.042	1.000	0.042	0.052	1.000	0.052
		Sys Arrivals/Hr =	Avg Serviced/Hr =	Sys-wide Rho =	Sys Arrivals/Hr =	Avg Serviced/Hr =	Sys-wide Rho =	Sys Arrivals/Hr =	Avg Serviced/Hr =	Sys-wide Rho =
		6.969	14.867	***0.469***	9.292	14.867	***0.625***	11.615	14.867	***0.781***

Scenario 1: System-Wide Analysis

The output under Scenario 1 conditions for average system queue size (Lq), average system size (L), average system queue wait time (Wq), and average system sojourn time (W) are displayed in Figure 8.

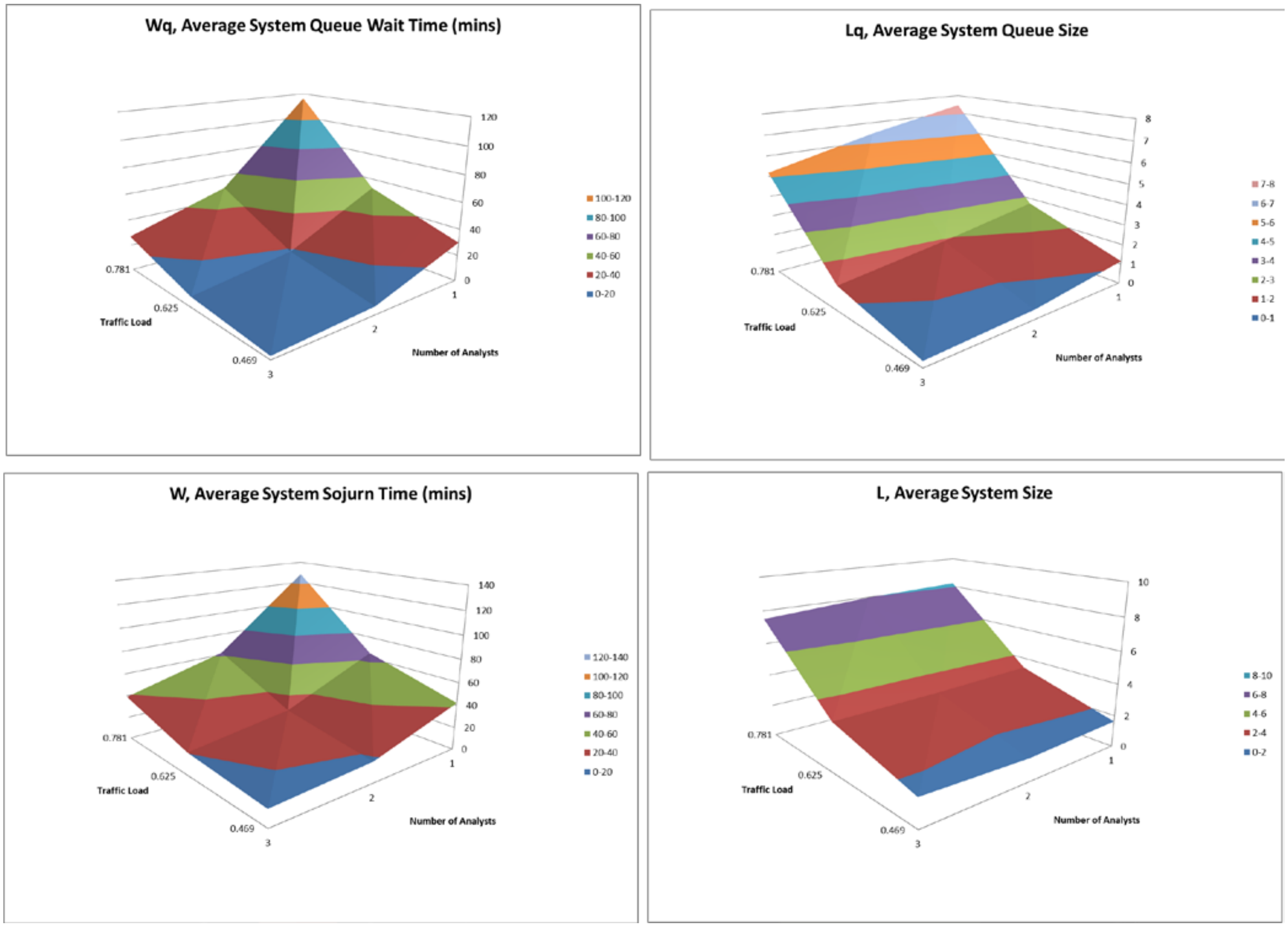


Figure 8. System response in terms of Wq, Lq, W, and L with an increasing number of analysts under three different traffic loads.

Average system queue wait time (Wq) and average system sojourn time (W) decrease with an increasing number of analysts, across all traffic loads. The steepest gains in performance were seen at the heaviest tested traffic condition (ρ=.781), with a 43% decrease in Wq (from 116.1 to 49.8 minutes) moving from one (1) to two (2) analysts and a 53% decrease in Wq (from 49.8 minutes to 26.8) when moving from two (2) to three (3) analysts.

Interestingly, Lq and L were relatively unaffected by the number of analysts in the system, instead tracking primarily with increasing traffic load. The sensitivity to traffic load might be expected since an increase in alert arrivals at the same servicing capacity results in more total alerts in the system at a given moment. Lq and L do decrease slightly with the number of analysts, and Lq appears to be more sensitive to the number of analysts than L. At ρ=.469, moving from one (1) to three (3) analysts decreased Lq by 74% (1.15 to .30 alerts), while L remained relatively stable at about 1.7 alerts. It should be noted that the sensitivity of Lq to the number of analysts decreased with increasing traffic load. At ρ=.781, Lq decreased by just 30% moving from one (1) to three (3) analysts (7.4 to 5.2 alerts), with L again experiencing very little movement, changing from 8.2 to just 7.5 alerts.

For the system as a whole, using a single analyst introduces a bottleneck which drives up all four response variables Wq, W, Lq, and L. Performance improvements from the employment of additional analysts stem from the creation of alternative paths through the system which alleviate bottleneck conditions. For instance, under the tested parameters, a single priority 1 alert takes an average of 3 hours to service. Therefore, in a single analyst system, when one of these alerts arrives, all other items in or entering the queue experience delays. The addition of more analysts provides alternative paths through the system for other alerts in the queue. It is unlikely that all analysts will be working priority 1 alerts at the same time due to their infrequent arrival rate.

Scenario 1: Analysis by Priority Level

In this section, the same four response variables (Wq, W, Lq, and L) are discussed, but this time broken up by priority level to compare them to the system-wide responses discussed above.

Figure 9 shows the response variables for Priority 3 alerts, which can be seen to track very closely to the system-wide results. The reason for this is the frequent arrival rate and short service times of Priority 3 alerts, which makes them a frequent occurrence in the system. Their sheer frequency skews the system-wide results in their favor.

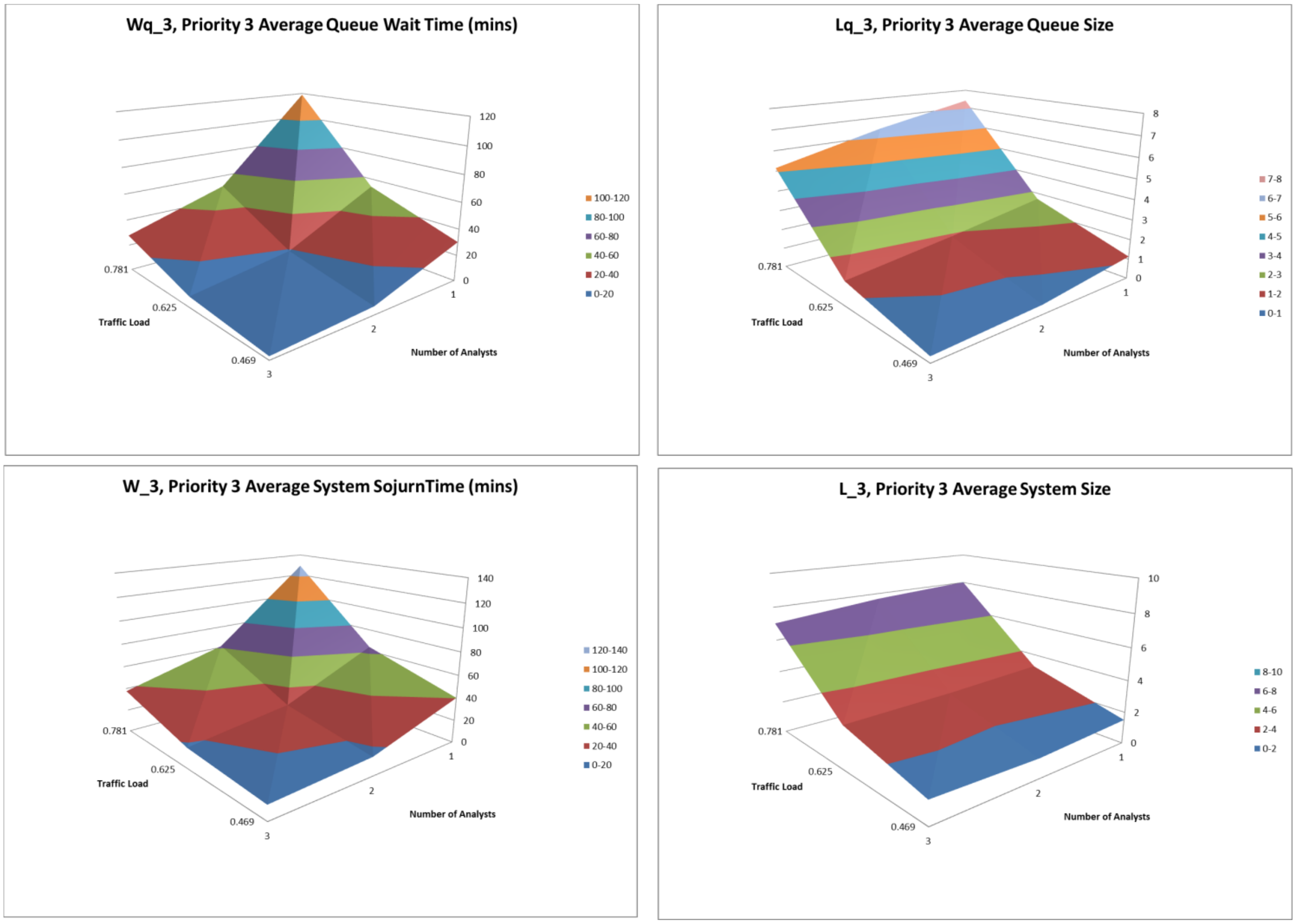


Figure 9. The four response variables (Wq, W, Lq, and L) for Priority 3 (low) alerts under Scenario 1 conditions. They track very closely to the system-wide results.

The situation is drastically different, however, when looking at the response of Priority 2 and Priority 1 alerts, shown in Figure 10 and Figure 11. The characteristic responses of Priority 1 alerts will be discussed specifically, but the same conclusions can be extended to Priority 2 results as well, since they demonstrate similar trends.

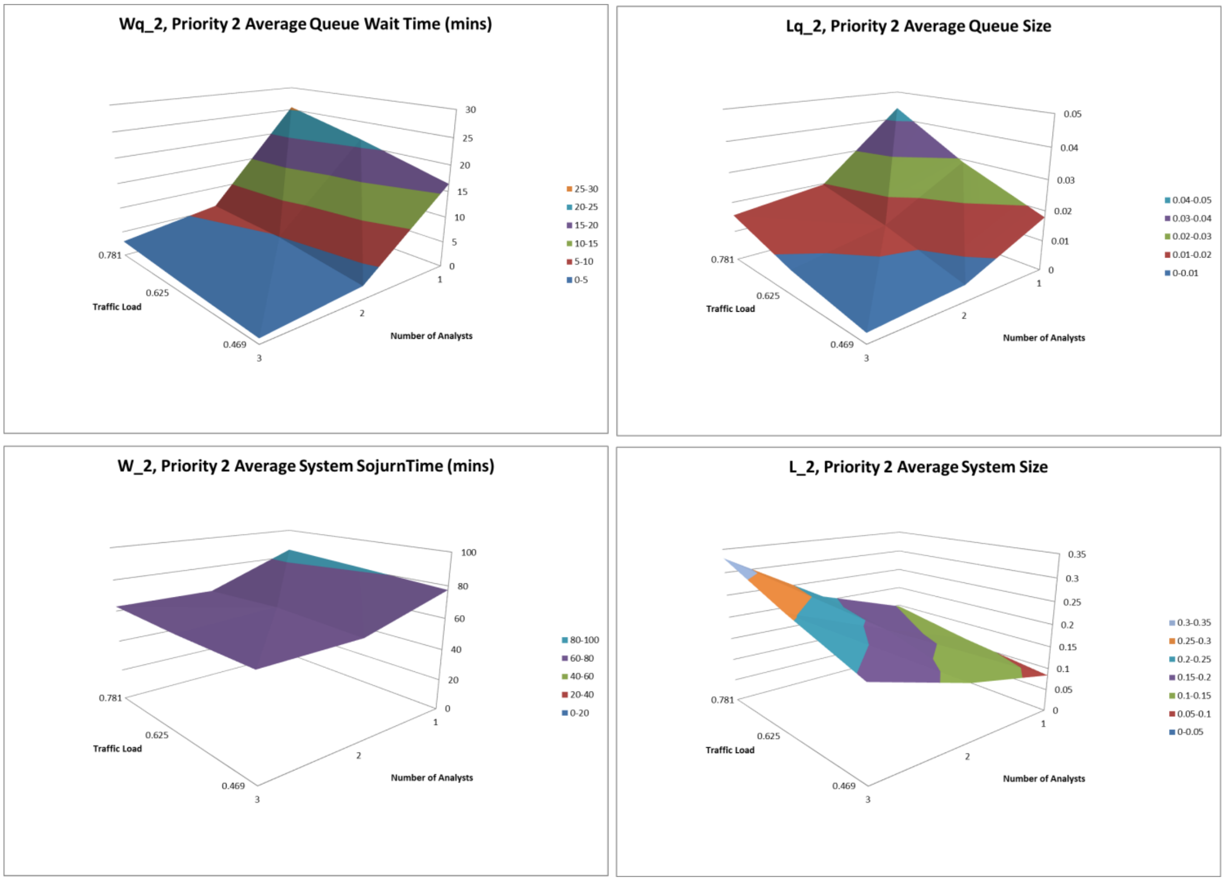


Figure 10. The four response variables (Wq, W, Lq, and L) for Priority 2 (mid) alerts under Scenario 1 conditions.

First, when looking at the average queue wait time for Priority 1 alerts, Wq_1, it is clear that Priority 1 alerts experience significantly reduced wait times compared to the system average. At ρ=.781 with one (1) analyst, Priority 1 alerts must only wait on average 21 minutes in the queue, compared to 116 minutes for the system. For the three (3) analyst case, Wq_1 drops to just 3 minutes, compared to nearly 27 minutes for the system. This stems from the rarity of Priority 1 alerts. The number of them expected to be in the queue, Lq_1, is extremely low (<<1) regardless of the number of analysts or traffic load. The net result is that, with multiple analysts, Priority 1 alerts will almost always be the very first item in the queue upon arrival, and must only wait for an analyst to finish their current alert before being pulled from the queue and receiving service.

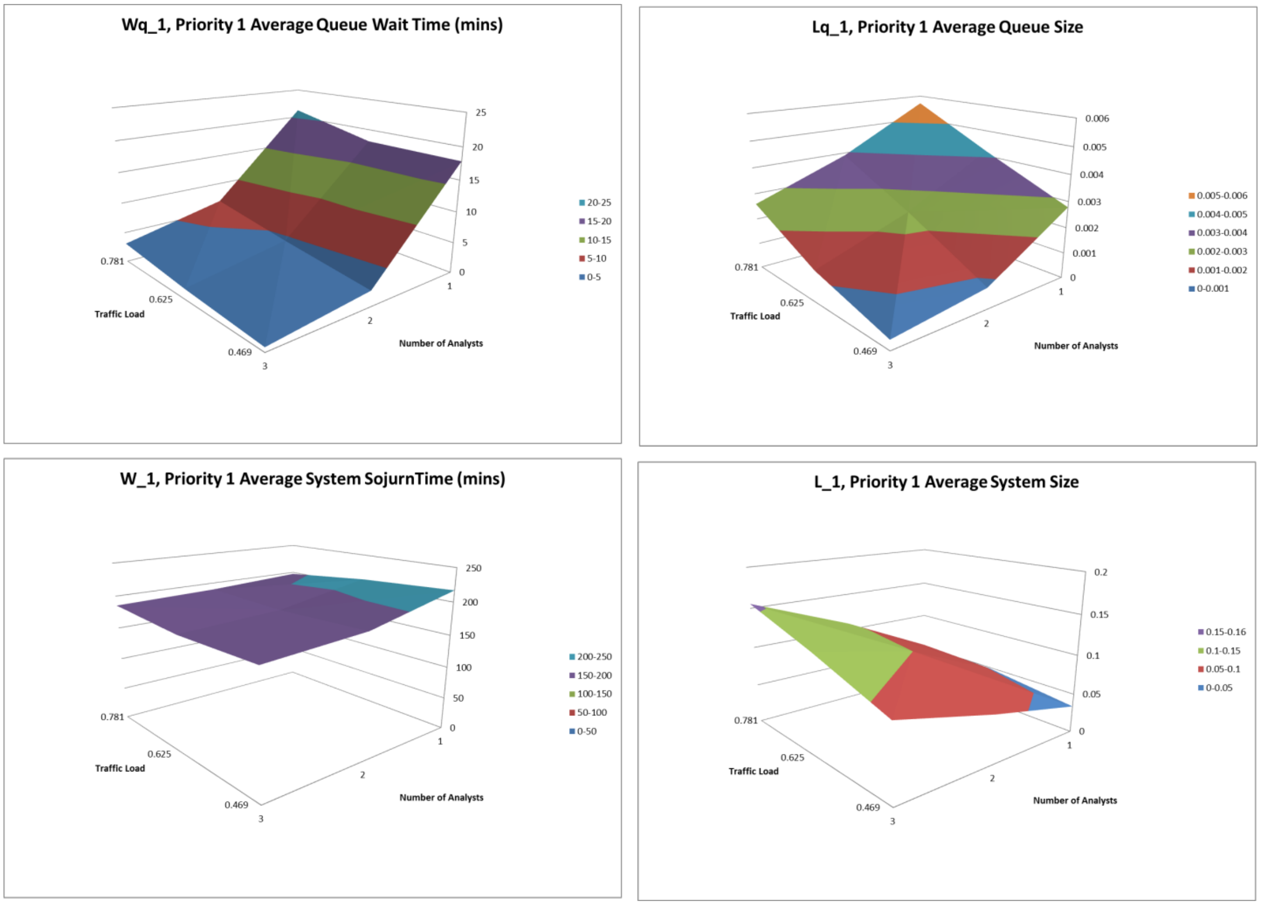


Figure 11. The four response variables (Wq, W, Lq, and L) for Priority 1 (high) alerts under Scenario 1 conditions.

As with Lq_1, L_1 is almost zero due to the infrequent occurrence of Priority 1 alerts, though it does increase slightly for an increasing number of analysts and increasing traffic load. This is simply due to the increased number of arrivals and increased probability that an analyst may be working a Priority 1 alert.

Interestingly, the average Priority 1 sojourn time, W_1, is nearly independent of both traffic load and number of analysts. This is driven by the short queue wait time for Priority 1 alerts, which means their time spent in the system is determined almost entirely by their average service time of 180 minutes.

Scenario 2: Varying the Number of Priority Levels

In the second scenario the focus was on determining how the alert priority scheme affected the response variables, both at the system level and by individual priority levels. Like Scenario 1, each response variable in Scenario 2 was examined under three different traffic loads, ρ=.469, .625, and .781.

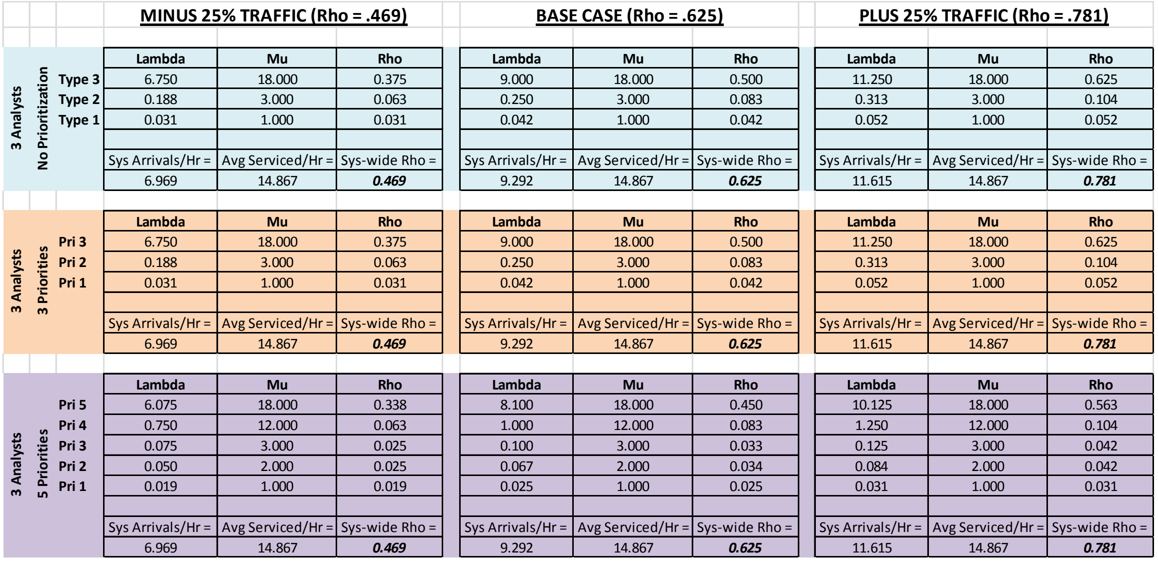
The 3-analyst case with 3 priority levels from Scenario 1 was taken as the base case for Scenario 2. Three analysts were used due to their demonstrated performance advantages in Scenario 1. Then, the base case was transposed into similar 1-priority and 5-priority cases.

Care had to be taken when forming the 1-priority level case. To give the illusion of a single priority level while maintaining a distribution of arrival rates and service times on par with those from Scenario 1, there had to be alerts with different properties (i.e. different arrival rates and service times) but they had to enter the queue without priority. This had to be done by tweaking the model code slightly. The tweaked model allowed for the definition of multiple "priorities", called Types in Table 6, with their own λ and μ values, but handled every Type of alert the same. In other words, all alerts had to start their journey through the system from the back of the queue.

Creating the 5-priority case was more straightforward. The arrival and service rates of the priority 1, 2, and 3 alerts from the 3-priority case were given to priority levels 1, 3, and 5 in the 5-priority case, respectively. Then, level 2 was given a new value for μ to fall between the values for levels 1 and 3. Level 4 was handled similarly. The arrival rates, λ, for all priority levels were adjusted as needed to ensure the appropriate overall system traffic densities were maintained.

A summary of Scenario 2 parameters is shown in Table 6.

Table 6. Summary of the model parameters used to generate each of the nine cases in Scenario 2.



			MINUS 25% TRAFFIC (Rho = .469)			BASE CASE (Rho = .625)			PLUS 25% TRAFFIC (Rho = .781)		
3 Analysts	No Prioritization		Lambda	Mu	Rho	Lambda	Mu	Rho	Lambda	Mu	Rho
		Type 3	6.750	18.000	0.375	9.000	18.000	0.500	11.250	18.000	0.625
		Type 2	0.188	3.000	0.063	0.250	3.000	0.083	0.313	3.000	0.104
		Type 1	0.031	1.000	0.031	0.042	1.000	0.042	0.052	1.000	0.052
			Sys Arrivals/Hr =	Avg Serviced/Hr =	Sys-wide Rho =	Sys Arrivals/Hr =	Avg Serviced/Hr =	Sys-wide Rho =	Sys Arrivals/Hr =	Avg Serviced/Hr =	Sys-wide Rho =
			6.969	14.867	***0.469***	9.292	14.867	***0.625***	11.615	14.867	***0.781***
3 Analysts	3 Priorities		Lambda	Mu	Rho	Lambda	Mu	Rho	Lambda	Mu	Rho
		Pri 3	6.750	18.000	0.375	9.000	18.000	0.500	11.250	18.000	0.625
		Pri 2	0.188	3.000	0.063	0.250	3.000	0.083	0.313	3.000	0.104
		Pri 1	0.031	1.000	0.031	0.042	1.000	0.042	0.052	1.000	0.052
			Sys Arrivals/Hr =	Avg Serviced/Hr =	Sys-wide Rho =	Sys Arrivals/Hr =	Avg Serviced/Hr =	Sys-wide Rho =	Sys Arrivals/Hr =	Avg Serviced/Hr =	Sys-wide Rho =
			6.969	14.867	***0.469***	9.292	14.867	***0.625***	11.615	14.867	***0.781***
3 Analysts	5 Priorities		Lambda	Mu	Rho	Lambda	Mu	Rho	Lambda	Mu	Rho
		Pri 5	6.075	18.000	0.338	8.100	18.000	0.450	10.125	18.000	0.563
		Pri 4	0.750	12.000	0.063	1.000	12.000	0.083	1.250	12.000	0.104
		Pri 3	0.075	3.000	0.025	0.100	3.000	0.033	0.125	3.000	0.042
		Pri 2	0.050	2.000	0.025	0.067	2.000	0.034	0.084	2.000	0.042
		Pri 1	0.019	1.000	0.019	0.025	1.000	0.025	0.031	1.000	0.031
			Sys Arrivals/Hr =	Avg Serviced/Hr =	Sys-wide Rho =	Sys Arrivals/Hr =	Avg Serviced/Hr =	Sys-wide Rho =	Sys Arrivals/Hr =	Avg Serviced/Hr =	Sys-wide Rho =
			6.969	14.867	***0.469***	9.292	14.867	***0.625***	11.615	14.867	***0.781***

Scenario 2: System-Wide Analysis

The system-wide results for each of the response variables are shown in Figure 12, for varying traffic load and numbers of priority levels.

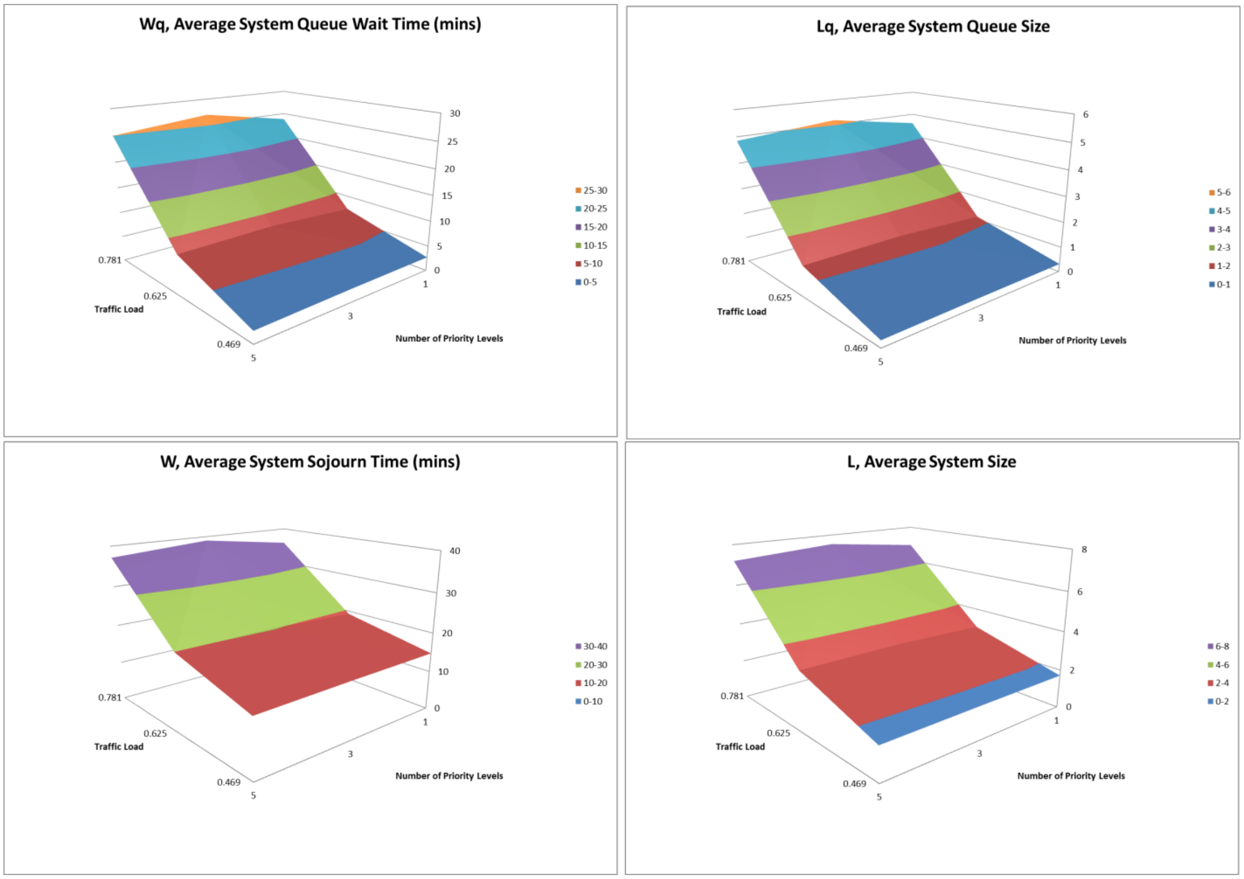


Figure 12. The system-wide response variables for the nine cases in Scenario 2.

When viewed from a system perspective, it would appear that the number of priority levels has no effect on Wq. W, Lq, or L. The response variable values remain constant as the number of priority levels changes. The only variation seen is with respect to traffic load, increasing non-linearly as traffic load increases.

While at first it may seem strange, this result makes sense if one remembers that only system-wide performance is being shown. The priority scheme is really just an internal feature of the system that allows some types of alerts to spend less time in the queue than others (we hope), but it does not affect the performance of the system as a whole. The next section, however, reveals that the situation “under the hood” is more complex than what the system-wide results depict.

Scenario 2: Analysis by Priority Level

The changes in the response variables by traffic load and individual priority levels are shown in Figure 13 for the 3-priority case and Figure 14 for the 5-priority case.

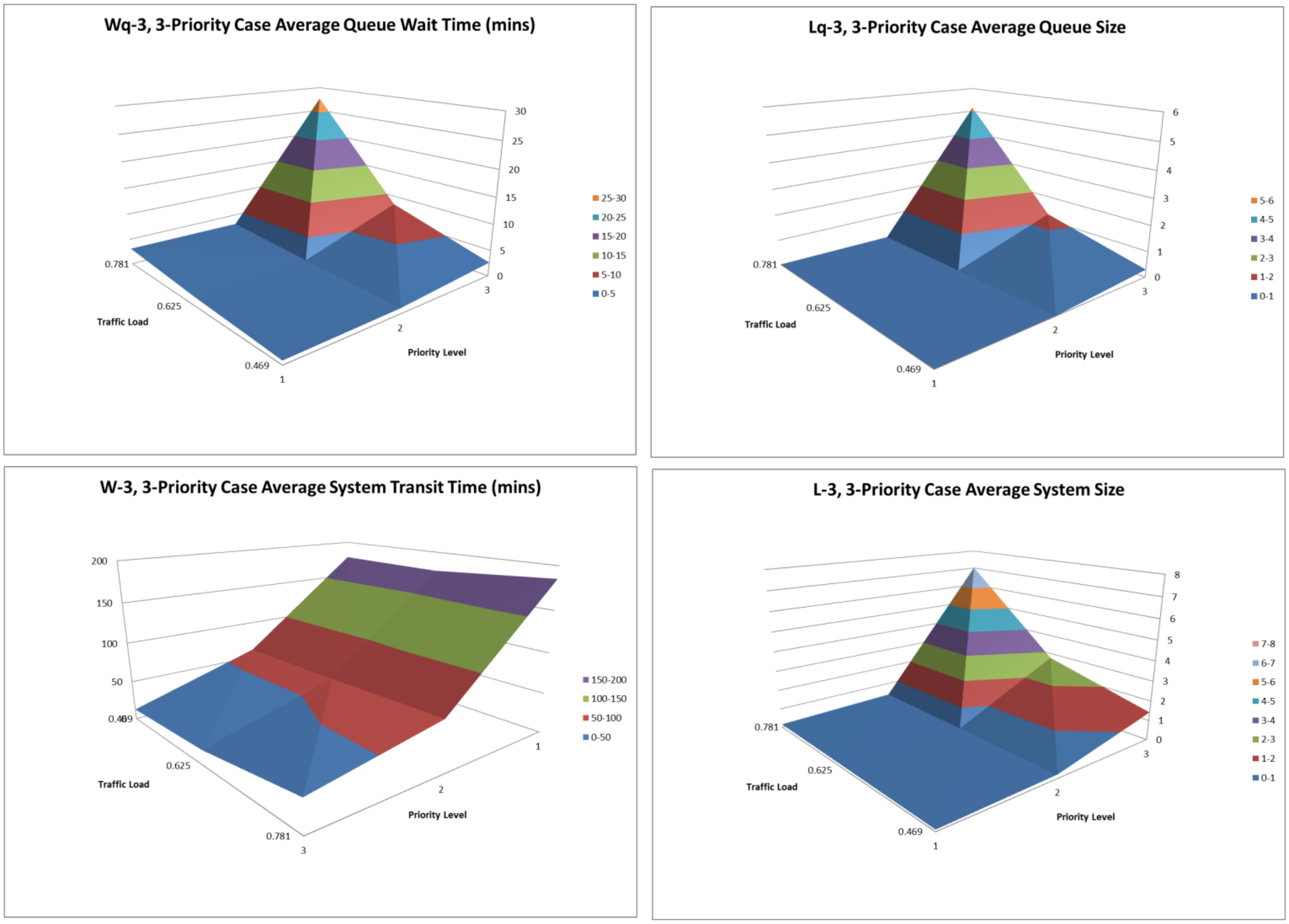


Figure 13. Response variables for the 3-priority case, broken out by individual priority level across three traffic conditions.

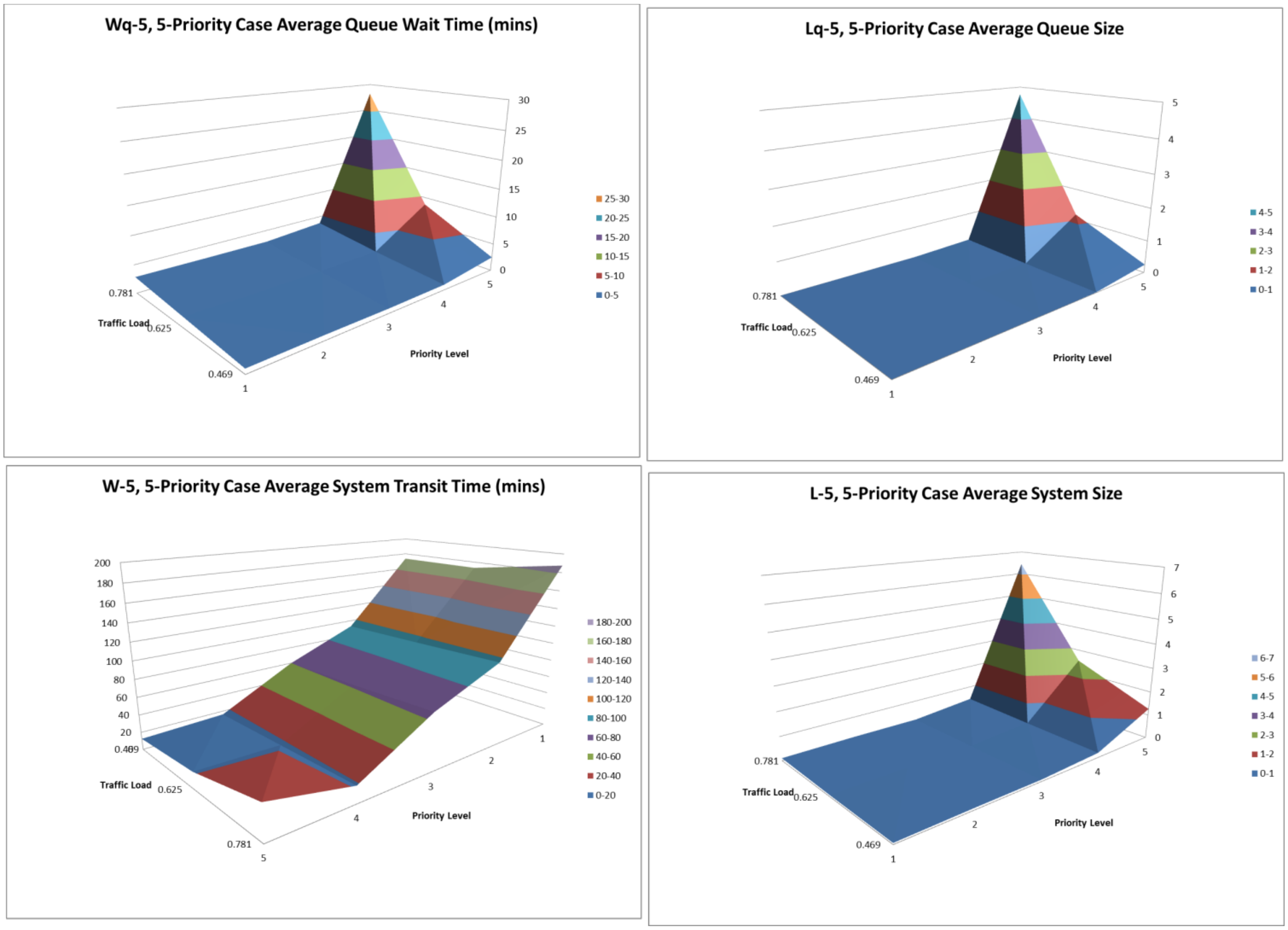


Figure 14. Response variables for the 5-priority case, broken out by individual priority level across three traffic conditions.

It is clear that in both the 3- and 5-priority cases, the lowest priority alerts (either level 3 or 5) suffer the most in terms of queue wait times, their numbers in the queue, and the total number in the system. This is expected, since the lowest levels must wait for the higher priorities to leave the system before they can be serviced. More interestingly, the queue wait times for the lowest level alerts are not appreciably impacted regardless of whether there are 2 or 4 levels of higher priority alerts above them. In both the 3- and 5-priority cases, Wq for levels 3 and 5 was about 28 minutes at ρ=.781.

But what about the high priority alerts? Does refining the shred out of priority levels push the highest priority alerts through the system any faster? The modeling results from Scenario 2 suggest the answer is no. The queue wait time for a level-1 alert under the 3-priority case, at ρ=.781, is right at 3 minutes. This wait time only decreases to 2.86 minutes under the 5-priority case. In fact, with ρ=.625, the queue wait time for a level-1 alert actually increases from 1.68 to 1.87 minutes. The explanation for this phenomenon has to do with the way the priory schemes were designed. The addition of more priority levels also introduced intermediate service times. Whereas with the 3-priority case most alerts were analyzed in an average of 10 minutes and a very small amount took 1 or 3 hours on average to service, the 5 priority case introduced a new 15 minute service time (μ=4) and a 1.5-hour service time (μ=2/3), so a greater percentage of all alert arrivals took, on average, more time to be serviced. This means when a level-1 alert arrived into the queue, even though it went straight to the front, it could expect to wait slightly longer in the queue since there were greater odds that the alert being serviced was not a level 5, but rather some slightly higher level with a higher service time than 10 minutes.

Regarding average system sojourn time, W, it can be shown that as the number of priority levels increases, the highest priority levels are able to come progressively closer to attaining their lower limit on system sojourn time, which is equal to their average service time, $1/\mu$. This is due to their decreased time spent waiting in the queue, so their total time in the system is limited only by their service time. As an example, at ρ=.781 and with 3 priority levels, the average sojourn time for a level-3 alert is 37.6 minutes, whereas the average service time is 10 minutes, 3.76 times longer. Under the same conditions, a level-1 alert has a sojourn time of 185.4 minutes, whereas the average service time is 180 minutes, a very minor difference.

Finally, the flat, blue planes in Figure 12 and Figure 13 for Wq, Lq, and L all represent the fact that there are very minor differences between the performances of priority levels 1 and 2 in the 3-priority case and levels 1-4 in the 5-priority case. Unless sufficient justification can be offered as to why a finer priority shred out is warranted, **the practical benefits of prioritization (i.e. near zero queue wait times for the most important alerts) can be realized with a simple 3-priority scheme.**

Summary

In this section, two scenarios were utilized to shed light on how network operations centers can make decisions about the key alert management variables under their control: the number of network analysts employed and the alert prioritization scheme used.

A single network analyst dramatically increases the potential for a system bottleneck, which can drive queue size and wait times up to unacceptable levels, potentially putting the network at risk. The addition of even one analyst provides an alternate route around the bottleneck, allowing for alerts of all priorities to reach an analysts screen in a short enough time frame to remain relevant.

Priority levels can ensure the most important or dangerous alerts reach an analysts screen quickly, but at the cost of increasing the amount of time the lowest level priority must wait in the queue. Too many priority levels may do little to add value to the alert management process, as the performance difference between the *n*-th priority level and *n*-1 are minimal and become even less relevant the larger *n* becomes.

Network operation centers should look at the tradeoffs between the number of analysts employed and the priority scheme utilized to make decisions that are suitable for their specific network environment.

V. Conclusions and Recommendations

Summary of Research

The core investigative question proposed in Chapter 1 was, given varying amounts of traffic density, how does the queuing system performance respond to varying numbers of analysts and priority levels? From this question and the analysis conducted in Chapter 4, several conclusions can be drawn.

First, even at the same traffic density, increasing the number of analysts decreases queue wait times and system sojourn time at the system level and across all individual priority levels. On the contrary, at a given traffic density, queue size and system size are largely independent of the number of analysts employed across all priority levels.

The significant performance gains from the use of any number of analysts greater than one is due to the reduced potential for system bottlenecks to occur, as more analysts mean alternative paths through the system when an analysts is occupied an alert requiring a long service time.

When the number of priority levels varies, it was found they have no effect on the system-wide performance metrics. Additionally, when the number of priority levels was increased above 3, the lowest-priority alerts were still shown to perform similarly to the system as a whole, while the additional shred out of higher priority alerts did little to push them through the system faster.

It was determined that the practical benefits of prioritization can be realized with a simple 3-priority scheme.

Study Limitations

Queuing Theory is useful for looking at stable, ongoing processes where tasks or customers arrive and are serviced at a given rate. In this research, the use of a human analyst as a server introduces some amount of uncertainty. Human analysts will not work continually like machines nor go straight from one customer to another like a grocery store cashier. They will take breaks and likely have other official duties that pull them away from their desks for some amount of time. Some leeway was added to account for this in the scenario assumptions about service times. Still, it should be acknowledged that the process of analyzing IDS alerts will have more variance than, say, people waiting in line for an amusement park ride or IP packets queuing up at a network router.

The assumptions regarding alert inter-arrival times were estimations based on the author's own experiences. No network data was collected. Data about alert generation rates from a live network would add more realism to the model. Fortunately, were this data to become available it would very simple to plug the derived parameters for λ into the model and run a new scenario. Even if the packets were to shown to arrive following some distribution other than exponential, it would just be a matter of changing one line of code to implement a new arrival distribution.

Recommendations for Action

If the DoD is serious about Active Cyber Defense - the implementation and continuous improvement of live network monitoring technologies is essential. Any network operations center using IDS devices should carefully consider how to manage alert flow based on its specific operating environment. Queuing models offer an excellent approach to characterize alert flow and make decisions about IDS configuration (such as alert priority levels) and analyst tasking.

Recommendations for Future Research

There are numerous avenues where future research could be conducted.

Implement FCFS with Preemptive Priority Resume Service Discipline

The service discipline that would be most realistic for the alert queuing system would be First-Come First-Served with Preemptive Priority Resume. In this service discipline, alerts are serviced in order of priority first, and then arrival time. However, if an alert enters the queue of a higher priority than one currently being serviced by an analyst, the lower priority alert is paused, set-aside, and the analyst immediately begins working the higher priority alert. When the higher priority alert is complete, the analyst will first check to ensure no other alerts of higher priority than the paused job are in the queue, and then continue working the lower priority job.

This service discipline has complexities that are amplified as priority levels are added to the model. For instance, when a lower priority alert is paused, one must consider whether that alert should stay with the same analyst, but in a paused state, or go back into the queue for re-assignment at a later time. Also, as priority levels are added, the probability that lower priority jobs will be preempted rises, so the model must account for and track an increasing number of paused alerts.

Explore the Effects of Alert Aggregation

A technique exists to deal with the issue of alert flooding from IDS devices called *alert aggregation*. This technique recognizes that many alerts generated share the same root-cause. For example, an alert may be in place to flag packets that travel over a rarely used port. In the event a hundred packets travel over that port, it could result in a hundred different alerts being generated, but with only on root cause that requires attention. The effects of alert aggregation could reduce the number of analysts required to monitor traffic, but with a possible loss in information (Saad & Traore, 2011).

Estimate Analyst Service Time using Bank Teller Data

The task of a network analyst examining an IDS alert may not be so different from that of a bank teller servicing customers. In both cases a human is performing tasks from a queue that result in variable services times. For the bank teller, some customers will require straight forward services (a quick deposit or cashing a check) while others will have questions or other non-routine issues that the teller must address. The analyst will experience a similar scenario as some alerts appear often enough to be deemed routine (but may still may have security value) while others require deeper investigation. Perhaps the biggest difference between the two is the bank teller scenario involves a human on both sides of the service interaction, whereas the analyst only one. Cogdill and Monticino, from University of North Texas, performed interesting analysis on data acquired from several branches of a regional bank chain regarding teller service times. They found the empirical data to more closely fit a log-normal distribution than an exponential (Cogdill & Monticino, 2007).

Collect Empirical IDS Data from Live Networks

The use of the exponential distribution to model alert inter-arrival times and service times in this paper is not based on actual network data. If the model were to be used to make decisions concerning a real world network operations center, it would be prudent to gather long-term data about actual alert inter-arrival times, which would likely vary based on time of day, day of the week, and so on. The same type of data should be collected about analyst service times. From the gathered empirical data, it would be possible to develop a distribution that more closely approaches the real world situation. Then, the model could be modified to allow different distributions for say, night or weekend or holiday operations, which may require differing numbers of analysts or even dictate the use of a different alert prioritization scheme.

Significance of Research

As the DoD network defense posture migrates from a passive filter approach to one of active, real-time counter-measures, the role of the network "operations center" must live up to its name and evolve beyond that of IT equipment manager. To do so, new operating concepts and analytical tools must be researched and applied. The application of Queuing Theory to the management of Intrusion Detection Systems (IDS) offers a powerful tool for the use of DoD network administrators in implementing an effective active cyber defense capability. It can be used to offer insight into efficient resourcing plans and help administrators develop effective device configurations. Though a small part of a comprehensive active cyber defense architecture, a well-honed IDS operated by proficient analysts (an end to which this research suggests a means) forms the foundation for implementing more advanced defense measures.

Appendix A

MATLAB Model Code

Below is the MATLAB code for the multi-server, multi-priority, first-come first-served queuing theory model used in this research. The code is divided into four parts: the "main" script which orchestrates the model, an Alert class, a Queue class, and an Analyst class. The Alert and Queue classes were modified from code written by Dimitri Shvorob (dimitri.shvorob@gmail.com), two whom all credit for the "queuing" functionality of the model should be given. Comments that appear in the coding of these classes are additions by the author. The Analyst class and main script are entirely the author's own work.

This is very much raw, working code. The author has attempted to clean it up for presentation and re-use, but there may be some lingering variables or class properties from "experimentation" during coding which have not been removed.

The "Main" Script (MMcFCFSwithPriority.m)

This is the main orchestrating function of the model. Here, the user can set the parameters for the simulation. This code will create instances from the Alert, Queue, and Analyst classes to orchestrate a full model. The reader may notice that the code generates a Queue object for every priority level. This does not mean the model uses multiple queues, but rather that each priority level is broken out into its own "virtual" queue to facilitate easier handling from a coding perspective. For example, the "real" queue may look like this, 5, 5, 4, 3, 3, 3, 2, 2, 1, 1, 1, 1, 1

But the model breaks this single queue into five virtual queues that look like this,

5, 5

4

3, 3, 3

2, 2

1, 1, 1, 1, 1

The net effect on model performance is the same, since alerts are still served in the same order based on priority.

Also of note, the code classifies alert priorities backwards from what is presented in this paper. For instance, if there are k=5 priority levels, a priority level of 5 is considered the highest priority and 1 the lowest. This convention is unfortunately a bit confusing, but could not be altered without significant re-writing of the code.

Finally, the code variables “lambda” and “mu” are really the values for 1/λ and 1/μ. So an alert with arrival rate λ=4 (4 alerts per hour) has a code “lambda” value of 1/4 (1 hour per 4 alert arrivals, or 15 minutes between alerts).

```
clear
clc

%%%% THESE ARE THE EDITABLE VARIABLES  %%%%%%%%%%%%%%%%

%set length of simulation, in # of alert arrivals
sim_length = 100000;

%set number of analysts
num_analysts = 3;

%set number of priority levels
k = 3;
```

```
% set arrival time parameters (must have a lambda value for each Alert
% priority level, lowest to highest, in a row vector [x, y, z...])
lambda = [1/4, 12, 72];

% set service time parameter (must have a mu value for each Alert
% priority level, lowest to highest, in a row vector [x, y, z...])
mu = [1/6, 1, 3];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% create a Queue of type Alert for each priority level
q = cell(1 , k);
for i = 1:k
    q{1, i} = Queue('Alert');
end

% create Analyst(s) at simulation start
a(1, num_analysts) = Analyst();


%setup the Alert arrival times for each priority level
arrival_times = zeros(sim_length, k);
avg_arrival_time = zeros(1, k);
cumulative_arrival_times = zeros(sim_length, k);
for i = 1:k
    arrival_times(:, i) = round(60 * exprnd(lambda(1, i), 1,
sim_length));
    avg_arrival_time(1, i) = mean(arrival_times(:, i));
    cumulative_arrival_times(:, i) = cumsum(arrival_times(:, i));
end

%make sure first arrival time for all priorities is not "zero"
for i = 1:k
    if cumulative_arrival_times(1,i) == 0;
        cumulative_arrival_times(1,i) = 1;
    end
end

%setup the service time array
service_times = zeros(sim_length, k);
avg_service_time = zeros(1, k);
for i = 1:k
    service_times(:, i) = round(60 * exprnd(mu(1, i), 1, sim_length));
    avg_service_time(1, i) = mean(service_times(:, i));
end

%create element counters for arrival and service time arrays, to track
when
%each element has been used
cumulative_arrival_element_counter(1, :) = ones(1, k);
service_times_element_counter(1, :) = ones(1, k);

%determine the last "minute" in simulation time, will stop
```

```
%1 time step less than longest cumulative arrival array to prevent an
indice
%overrun problem in the main "for" loop
end_of_sim_time = min(cumulative_arrival_times(sim_length, :))-1 ;

%preallocate an array to store the queue size & system size at each
time step
q_sizes = zeros(end_of_sim_time, k);
sys_sizes = zeros(end_of_sim_time, k);

%preallocate arrays to store the queue & system wait times, each time
an alert is
%polled from the queue or finished
q_wait_times = zeros(sim_length, k);
sys_wait_times = zeros(sim_length, k);

%create counter variables to track which element the queue and system
%times should be placed in
q_wait_counter(1, :) = ones(1, k);
sys_wait_counter(1, :) = ones(1, k);

%This is the "guts" of the model, where alerts enter the queue, are
%assigned a server, and all data about queue lengths and wait times are
%tracked
for sim_time = 1:end_of_sim_time

    %check for alert arrivals
    %   if true, place them in the queue -- otherwise don't do anything
    %   (this has to be a while loop because sometimes the arrival time
is "0"
    %    and therefore two alerts arrive at the same moment in sim
time)
    for j = k:-1:1
        while
cumulative_arrival_times(cumulative_arrival_element_counter(1, j), j)
== sim_time
            q{1, j}.offer(Alert(j, sim_time));
            cumulative_arrival_element_counter(1, j) =
cumulative_arrival_element_counter(1, j) + 1;
        end
    end



    %check to see if any Analyst has finished working
    %if true,
    %   calc amount of time alert was in system, clear the alert & make
the analyst available
    for i = 1:num_analysts
        if (a(1, i).get_available ~= 1) &&  (a(1,
i).alert.time_finished == sim_time)
            sys_wait_times(sys_wait_counter(1, a(1, i).alert.priority),
a(1, i).alert.priority) = sim_time - a(1, i).alert.time_arrived;
            sys_wait_counter(1, a(1, i).alert.priority) =
sys_wait_counter(1, a(1, i).alert.priority) + 1;
```

```
            a(1, i) = a(1, i).clear_alert;

            a(1, i) = a(1, i).make_available;
        end
    end

    %check to see that an Analyst is available & that Queue is not
empty
    %if true,
    %   give Analyst next Alert in the queue & save the time it was
polled
    %   make the Analyst busy
    %   store when the Alert will be finished
    for i = 1:num_analysts
        for j = k:-1:1
            while a(1, i).get_available == 1 && q{1, j}.isempty ~= 1
                a(1, i) = a(1, i).assign_alert(q{1, j}.poll, sim_time);

                q_wait_times(q_wait_counter(1, j), j) = a(1,
i).alert.time_polled - a(1, i).alert.time_arrived;
                q_wait_counter(1, j) = q_wait_counter(1, j) + 1;

                a(1, i) = a(1, i).make_busy;

                a(1, i).alert.time_finished = sim_time +
service_times(service_times_element_counter(1, j), j);
                service_times_element_counter(1, j) =
service_times_element_counter(1, j) + 1;

                %check to see if any Analyst has finished working
                %if true,
                %   calc amount of time alert was in system, clear the
alert & make the analyst available
                if (a(1, i).get_available ~= 1) &&  (a(1,
i).alert.time_finished == sim_time)
                    sys_wait_times(sys_wait_counter(1, a(1,
i).alert.priority), a(1, i).alert.priority) = sim_time - a(1,
i).alert.time_arrived;
                    sys_wait_counter(1, a(1, i).alert.priority) =
sys_wait_counter(1, a(1, i).alert.priority) + 1;

                    a(1, i) = a(1, i).clear_alert;

                    a(1, i) = a(1, i).make_available;
                end
            end
        end
    end


    %store the queue size at current simulation time
    for j = k:-1:1
```

```
       q_sizes(sim_time, j) = q{1, j}.size;
    end

    %store the system size at current simulation time
    alerts_in_service = zeros(1, k);
    for j = k:-1:1
        for i = 1:num_analysts
            if (a(1, i).get_available ~= 1) && (a(1, i).alert.priority
== j)
                alerts_in_service(1, j) = alerts_in_service(1, j) + 1;
            end
        end
        sys_sizes(sim_time, j) = q{1, j}.size + alerts_in_service(1,
j);
    end
end

%Calculate system-wide Lq, L, Wq, and W

Lq = zeros(1, k);
for j = k:-1:1
    Lq(1, j) = mean(q_sizes(:, j));
end
Lq_overall = sum(Lq)

L = zeros(1, k);
for j = k:-1:1
    L(1, j) = mean(sys_sizes(:, j));
end
L_overall = sum(L)

Wq = zeros(1, k);
for j = k:-1:1
    Wq(1, j) = mean(q_wait_times(1:q_wait_counter(1, j), j));
end
weighted_sum = 0;
for j = k:-1:1
    weighted_sum = weighted_sum + Wq(1, j)*q_wait_counter(1, j);
end
Wq_overall = weighted_sum / sum(q_wait_counter(:,:))

W = zeros(1, k);
for j = k:-1:1
    W(1, j) = mean(sys_wait_times(1:sys_wait_counter(1, j), j));
end
weighted_sum = 0;
for j = k:-1:1
    weighted_sum = weighted_sum + W(1, j)*sys_wait_counter(1, j);
end
W_overall = weighted_sum / sum(sys_wait_counter(:,:))
```

The Alert Class (Alert.m)

This class is used to construct Alert objects, which enter the queue and are serviced by analysts. The Queue class will call on the “eq” and “gt” functions within the Alert class to store incoming alerts into their proper queue position.

```
classdef Alert

    properties %(SetAccess = private)
        priority          %this property defines the priority (1, 2, 3,
etc.) of the alert
        time_arrived      %store the simulation time at which the alert
was introduced to the queue
        time_polled       %store the simulation time at which the alert
was polled from the queue
        time_finished     %store the simulation time at which the alert
will be finished
    end

    methods

        %Construction function for the Alert class
        function[obj] = Alert(x, t)
            obj.priority = x;
            obj.time_arrived = t;
            obj.time_polled = 0;
            obj.time_finished = 0;
        end

        %returns the priority level of an alert
        function[out] = get_priority(obj)
            out = obj.priority;
        end

        %compares the priority level of two Alert objects to determine
if
        %they are equal
        function[out] = eq(obj,obj2)
            if length(obj2) > 1
               throw(MException('Widget:eqMultiple','??? Cannot compare
to multiple elements at once.'))
            end
            out = strcmp(class(obj),class(obj2)) && obj.priority ==
obj2.priority;   % obj2 must be of the same class
        end

        %compares the priority level of two Alert objects to determine
if
        %the first object is higher priority than the second object
        function[out] = gt(obj,obj2)
```

```
            if length(obj2) > 1
               throw(MException('Widget:gtMultiple','??? Cannot compare
to multiple elements at once.'))
            end
            out = isa(obj2,'Alert') && obj.priority > obj2.priority;
% obj2 must be an Alert
        end

    end

end
```

The Queue Class (Queue.m)

This class creates a queue for holding Alert objects until they can be passed onto an Analyst for service.

```
classdef Queue < handle

   % Queue - strongly-typed Queue collection
   %
   % Properties:
   %
   %   Type (string)
   %
   % Methods:
   %
   %   Queue(type)
   %   display
   %   size
   %   isempty
   %   clear
   %   contains(obj)
   %   offer(obj)
   %   remove(obj)
   %   peek   - returns [] if queue is empty
   %   poll   - returns [] if queue is empty
   %   values - returns contents in a cell array
   %
   % Notes:
   %
   % Compatible classes must overload eq() for object-to-object
comparisons.
   %
   % Author: dimitri.shvorob@gmail.com, 3/15/09

   properties (GetAccess = protected, SetAccess = protected, Hidden =
true)
       Elements   %Elements are the "cells" that hold the queued
objects
   end
   properties (SetAccess = protected)
       Type    %stores the type of object the queue will be storing;
cannot
               %mix different type objects into a single queue
   end

   methods

       %Queue constructor function
       function[obj] = Queue(type)
           if ~ischar(type)
              throw(MException('Queue:constructorInvalidType','???
''type'' must be a valid class name.'))
           end
           obj.Elements = {};
           obj.Type = type;
       end

       %function to view the current state of the queue
       function disp(obj)
           disp([class(obj) '<' obj.Type '> (head on top)'])
           if ~obj.isempty
              for i = 1:obj.size
                  disp(obj.Elements{i})
              end
           else
              disp(['empty'])
           end
       end

       %output the current size (or length) of the queue
       function[out] = size(obj)
           out = length(obj.Elements);
       end

       %I had to modify the "values" function because sometimes it
output a
       %row vector and sometimes a column vector, which was
troublesome.
       %Now it always outputs a column vector.
       function[out] = values(obj)
           dummy = cell(obj.size, 1);
           for i = 1:obj.size
               dummy{i,1} = obj.Elements{i};
           end
           out = dummy;
       end

       %Checks to see if the queue is empty; 1 = yes, 0 = no
       function[out] = isempty(obj)
           out = obj.size == 0;
       end
       %clears an object from a particular element in the queue
       function[obj] = clear(obj)
           obj.Elements = {};
       end

       %Checks to see if a particular element contains a particular
object
       %- NOT USED IN THE QUEUING THEORY MODEL...but could be useful
       function[out] = contains(obj,e)
           out = false;
           for i = 1:obj.size
               if e == obj.Elements{i}
                  out = true;
                  break
               end
           end
       end

       %Places an object into the queue
       function[obj] = offer(obj,e)
           if length(e) > 1
              throw(MException('Queue:offerMultiple','??? Cannot offer
multiple elements at once.'))
           end
           if ~isa(e,obj.Type)
              throw(MException('Queue:offerInvalidType','??? Invalid
type.'))
           end
           if isempty(obj.Elements)
              obj.Elements = {e};
           else
              obj.Elements{end+1} = e;
           end
       end

       %Removes an object from the queue
       function[obj] = remove(obj,e)
           if length(e) > 1
              throw(MException('Queue:removeMultiple','??? Cannot
remove multiple elements at once.'))
           end
           if ~isa(e,obj.Type)
              throw(MException('Queue:removeInvalidType','??? Invalid
type.'))
           end
           if ~isempty(obj.Elements)
              k = [];
              for i = 1:obj.size
                  if e == obj.Elements{i}
                     k = [k i];  %#ok
                  end
              end
              if ~isempty(k)
                  obj.Elements(k) = [];
              end
           end
       end

       %Shows what the next item in the queue is, but leaves it in
place
       function[out] = peek(obj)
           if ~obj.isempty
               out = obj.Elements{1};
           else
               out = [];
           end
       end

       %Pulls the next item from the queue, and removes it from the
queue
       function[out] = poll(obj)
           if ~obj.isempty
               out = obj.Elements{1};
               obj.Elements(1) = [];
           else
               out = [];
           end
       end

   end

end
```

The Analyst Class (Analyst.m)

This class enables the creation of Analyst objects, which can take alerts that are pulled from the queue and service them.

```
% Class to serve as analysts
classdef Analyst

    properties
        available  %1 = available, 0 = busy
        alert      %holds the alert object the analyst is currently
servicing
    end

    methods

        %Construction function for the Analyst class
        function[obj] = Analyst()
            obj.available = 1;
            obj.alert = {};
        end
        %determine if Analyst is currently servicing an Alert
        function[out] = get_available(obj)
            out = obj.available;
        end

        %assigns an alert to an analyst
        function[obj] = assign_alert(obj, e, t)
            obj.alert = e;
            obj.alert.time_polled = t;  %store the time the alert was
given
                                        %to the analyst
        end


        %removes Alerts when the analyst finishes working on them
        function[obj] = clear_alert(obj)
            obj.alert = {};
        end

        %changes an Analysts Available property to "busy" when the
        %Analyst is assigned an alert
        function[obj] = make_busy(obj)
            obj.available = 0;
        end

        %changes an Analysts "Available" property to "available" when
the
        %Analyst completes an Alert
        function[obj] = make_available(obj)
            obj.available = 1;
        end

        %returns the priority of the current Alert the Analyst is
working
        function[out] = get_alert_priority(obj)
            out = obj.alert.priority;
        end

    end

end
```

Bibliography

Aziz, N. (2006). Intrusion Alert Correlation. MIMOS.

Blackwell, J. (2004). RAMIT - Rule-Based Alert Management Information Tool. The Florida State University.

Cogdill, T., & Monticino, M. (2007). Analysis of Teller Service Times in Retail Banks. *Case Studies in Business, Industry, and Government Statistics (CS-BIGS)*, 15-25.

Dept of Defense Chief Information Officer. (2010, August). *The DoDAF Architecture Framework Version 2.02*. Retrieved August 2014, from Dept of Defense Chief Information Officer: http://dodcio.defense.gov/TodayinCIO/DoDArchitectureFramework.aspx

Harchol-Balter, M., Osogami, T., Scheller-Wolf, A., & Wierman, A. (2005). Multi-server queueing systems with multiple priority classes. Department of Computer Science, Carnegie Mellon.

Meyer, R. (2008). Challenges of IDS in the Enterprise. SANS Institute.

Morin, B., Me, L., Debar, H., & Ducasse, M. (n.d.). M2D2: A Formal Data Model for IDS Alert Correlation. France Telecom R&D, Caen, France.

Rasch, G. (1963). The Poisson Process as a Model for a Diversity of Behavioral Phenomena. *International Congress of Psychology.* Washington D.C.

Raulerson, E. L., Hopkinson, K. M., & Laviers, K. R. (2014). A Framework to Facilitate Cyber Defense Situational Awareness. Air Force Institute of Technology.

Reed, D. A. (1995). *Introduction to Queueing Theory, ECE/CS 441 Notes*. Retrieved September 15, 2014, from University of Illinois: http://users.crhc.illinois.edu/nicol/ece541/slides/queueing.pdf

Roesch, M., & Green, C. (2014). *SNORT Users Manual.* Retrieved August 2014, from SNORT: http://manual.snort.org/

Saad, S., & Traore, I. (2011). A Semantic Analysis Approach to Manage IDS Alerts Flooding. *7th International Conference on Information Assurance and Security (IAS)* (pp. 156-161). Malacca: IEEE.

Schudel, G. (n.d.). *Bandwidth, Packets Per Second, and Other Network Performance Metrics*. Retrieved August 2014, from Cisco Security Intelligence Operations: http://www.cisco.com/web/about/security/intelligence/network_performance_metrics.html

Sigman, K. (n.d.). *Lecture Notes on Stochastic Modeling I.* Retrieved Oct 25, 2014, from Karl Sigman's Homepage: http://www.columbia.edu/~ks20/stochastic-I/stochastic-I-LL.pdf

Tedesco, G., & Aickelin, U. (n.d.). Strategic Alert Throttling for Intrusion Detection Systems. The University of Nottingham, United Kingdom.

Valeur, F., Vigna, G., Kruegel, C., & Kemmerer, R. A. (2004). A Comprehensive Approach to Intrusion Detection Alert Correlation. *IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING*.

Virtamo, J. (n.d.). 38.3143 Queuing Theory / Priority Queues.

Vita

Captain Christopher C. Olsen is a Developmental Engineer assigned to the 707th Intelligence, Surveillance, and Reconnaissance Group with duty at the National Security Agency, Fort Meade, Maryland. He currently serves as the 707th ISRG commander's Chief of Plans and Programs, where his primary focus is improving the integration of signals intelligence capabilities with air and space operations.

Captain Olsen graduated from Texas A&M University in 2008, earning his commission through ROTC. His initial duty was at the National Air and Space Intelligence Center at Wright-Patterson AFB, Ohio, where he served as a Space Employment Engineer. In 2011, Captain Olsen was assigned to the National Security Agency, where he was a Branch Chief leading a small engineering team in the procurement of advanced cryptologic and cyber defense technologies. Concurrently while serving as Branch Chief, Captain Olsen was also a Flight Commander for 35 airmen who provided linguistic and signals intelligence to numerous combatant commands and national agencies.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to an penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)* 26-12-2014	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From – To)* Jan 2014 – Dec 2014

4. TITLE AND SUBTITLE

Characterizing and Managing Intrusion Detection System (IDS) Alerts with Multi-Server/Multi-Priority Queuing Theory

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Olsen, Christopher C., Captain, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENV-MS-14-D-24

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Intentionally left blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A. APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

The DoD sets forth an objective to "employ an active cyber defense capability to prevent intrusions onto DoD networks and systems." Intrusion Detection Systems (IDS) are a critical part of network defense architectures, but their alerts can be difficult to manage. This research applies Queuing Theory to the management of IDS alerts, seeking to answer how analysts and priority schemes effect alert processing performance. To characterize the effect of these two variables on queue wait times, a MATLAB simulation was developed to allow parametric analysis under two scenarios. The first varies the number of analysts and the second varies the number of alert priority levels. Results indicate that two analysts bring about drastic improvements (a 41% decrease) in queue wait times (from 116.1 to 49.8 minutes) compared to a single analyst, due to the reduced potential for bottlenecks, with diminishing returns thereafter. In the second scenario, it was found that three priority levels are sufficient to realize the benefits of prioritization, and that a five level priority scheme did not result in shorter wait queue times for Priority 1 alerts. Queuing models offer an effective approach to make IDS resource decisions in keeping with DoD goals for Active Cyber Defense.

15. SUBJECT TERMS

Queuing Theory, Intrusion Detection Systems, Active Cyber Defense, Alerts, Priority Queues

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
					John M. Colombi, Ph.D, USAF
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(Include area code)*
U	U	U	UU	84	(937) 255-6565, ext 3347 John.Colombi@afit.edu

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18