DISTRIBUTED FRAMEWORK FOR DEPLOYING MACHINE LEARNING IN NETWORK MANAGEMENT AND SECURITY

By

John Alan Bivens II

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

Approved by the
Examining Committee:

_____
Dr. Boleslaw Szymanski, Thesis Adviser

_____
Dr. Mark Embrechts, Member

_____
Dr. Mark Goldberg, Member

_____
Dr. Shivkumar Kalyanaraman, Member

_____
Dr. Dinesh Verma, Member

Rensselaer Polytechnic Institute
Troy, New York

February 2003
(For Graduation May 2003)

DISTRIBUTED FRAMEWORK FOR DEPLOYING MACHINE LEARNING IN NETWORK MANAGEMENT AND SECURITY

By

John Alan Bivens II

An Abstract of a Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

The original of the complete thesis is on file
in the Rensselaer Polytechnic Institute Library

Examining Committee:

Dr. Boleslaw Szymanski, Thesis Adviser
Dr. Mark Embrechts, Member
Dr. Mark Goldberg, Member
Dr. Shivkumar Kalyanaraman, Member
Dr. Dinesh Verma, Member

Rensselaer Polytechnic Institute
Troy, New York

February 2003
(For Graduation May 2003)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

I, as well as this thesis, am a compilation of efforts from many people to help develop and guide me through the years. I would first like to thank my colleagues and fellow students who worked on many of these projects with me, including Joel Branch, Michelle Conway Breimer, Chi Yu Chan, Chris Cramer, Seth Freeman, Li Gao, Rashim Gupta, Taek Kyeun Lee, Ingo McLean, Chandrika Palagiri, Vivek Rao, Rasheda Smith, Sunil Uplaker, and Jerome White. These students were more than just colleagues, but they were dear friends whom I will always cherish. Support does not always come in the form of time in front of a machine, many of my friends that I had no opportunity to work with also played a part in my success, including Brandies Hill, Gemez Marshall, Lesley Mbogo, Roshawnna Scales, and Quincy Stokes.

I would like to give special thanks to David Kotfila who graciously made the CISCO router lab available to us during our many test runs. Without his help, much of this work would not have been possible. At this time I would also like to extend a very special thanks to three exceptional members of my thesis committee: Dr. Mark Embrechts, Dr. Boleslaw Szymanski, and Dr. Dinesh Verma for their un-tiring guidance and assistance. These three individuals personally taught and helped me every step along the way. Without their help I would certainly not be where I am today. Thank you.

Lastly I would like to thank Attorney John Bivens and Dr. Carmelita Bivens, my parents, for always understanding and supporting my many efforts. I would also like to thank Mrs. and Dr. Fletcher Jones, Maj Keecia Jones and Brian Jones, the family I have just recently entered, for their encouragement and support through this process.

Most of all, I would like to thank Mrs. Kimberley Bivens, my wife, whose smile alone is enough motivation to make any man succeed. Thank you for always being there, and providing a place of love and happiness for the last 5.5 years. This degree is as much yours as it is mine.

Thank You All. –Dr. John Alan Bivens II

ABSTRACT

As computer networks grow in frequency, size, and degree of segmentation, network management applications must not only provide efficiency, non-intrusiveness, and reliability, but also must be able to scale these characteristics over a wide range of architectures. In this thesis, we propose a distributed framework for network management middleware. The goal is to easily distribute functional agents to locations where they may carry on the actions of the management application closer to the managed node. In large networks with multiple managers, problems in a network usually draw attention and management traffic to the problem location. This added management traffic only exaggerates the problem. We show and quantify the benefits of the proposed distribution by implementing several real-time network managers using our distributed framework. We also propose and describe several management techniques, including congestion control and network parameter optimization, which use the distributed framework.

The effects of the agent-based distribution that we developed also enables the application of centrally managed but functionally distributed agents to fields where scalable, centralized management was not practical. One such field is Intrusion Detection. Currently, intrusion detection processes are individually installed and independently managed preventing large-scaled distributed detection. We propose and describe here many intrusion detection methods targeting both host-based and network-based attacks. Consistent across our work is the application of intelligence or learning techniques, such as Perceptron-based neural networks, self organizing maps, and genetic algorithms.

Our approaches strive to provide scalability. We study and provide a process for simulation-based scalability evaluation through application segmentation and its relationship to real networks. We have also formed a mathematical analysis of comparative scalability to analyze various application architectures. We compare our framework to traditional architectures of network management using all of these methods.

CHAPTER 1
Introduction

1.1 Network Management

In the growing world of networking, increasing emphasis is being placed upon speed, connectivity, and reliability of the networks. Network performance is vital to businesses for both intra-business operations as well as bringing products to consumers. Networking has played a part in changing the household, as the interconnectivity of families and friends has changed the way we communicate and seek information.

Network management plays an important part in this process in both proactive and reactive ways. Network problems disrupt services to those depending on this medium at costs up to thousands, or even millions, of dollars in all forms of commerce. Whether a company uses a network to receive information, to communicate within the company, or to actually conduct business and engage in commerce, it can be crippled when network problems prevent the network from operating at expected capacity. In addition to preserving methods of communication and commerce, network management also plays an important role in optimizing existing networks, thus supporting additional or new networking applications. As computer networks increase in size and level of segmentation, network management must focus on efficient, scalable data acquisition and processing. Companies are reluctant to continue to invest large amounts of capital into networking infrastructure and would rather get more out of the existing infrastructure through better management and new networking applications.

In this thesis, we propose and study a middle-ware to provide for scalable, functionally distributed network managers. There are many fields of network management but this thesis focuses on the following areas, explained briefly in the following subsections and extensively in the following chapters.

- Monitoring and Modeling

- Information Security
- Congestion Control
- Parameter Optimization

1.1.1 Monitoring and Modeling

The monitoring and modeling methods usually include collecting data about the activities of the network through some of the network's components. Typical applications which use these techniques would provide information to administrators as to the status of a particular device or activity of a certain link. Management applications may set thresholds on monitored values to alert the appropriate people when the network may be in or about to enter an undesirable state.

Monitoring usually involves an agent on the network device which calculates and stores certain pieces of information for retrieval by a manager [103]. The agent running on the network device usually only calculates and stores basic pieces of information, keeping most of the computational and storage resources available for the original purpose of the networking device (typically a router, switch, host, or other network accessory).

Modeling applications many times use multiple readings of this monitoring information to model the states of the network over time. This model may be used for forecasting, problem investigation, or in real-time to assist in efficient data delivery.

Both monitoring and modeling applications usually require a great deal of data from these networking devices on a regular basis. These applications are many times the first line of defense against network failures, as well as an integral part of failure investigation.

1.1.2 Information Security

Information security is a large area consisting of preventive measures (firewalls, encryption, etc.) and detection methods (intrusion detection). Of these two areas, we focus on intrusion detection. Today's corporations and educational institutions are particularly vulnerable to network attacks. These attacks cause disruptions

of service, damage to existing systems, and in some cases, unauthorized access to sensitive information. Attacks addressed by intrusion detection are usually divided into two groups consisting of attacks from within the network (host-based) and attacks from outside of the network (network-based).

1.1.2.1 Host-Based vs. Network-Based Intrusion Detection

To detect host-based attacks, per-host logging is performed, like an audit system engaged on a host that records all of the system calls that a particular user initiates. This log of system calls is a fine record of a user's activities and can be used to detect suspicious behavior. In particular, this type of data is especially useful in detecting masquerading attacks. In these cases, when a user's account is compromised by an attacker, the attacker's goal is to continue undetected use of the account. Therefore, the attacker causes no direct damage to the user and tries not to draw attention to himself.

To detect network-based attacks, a traffic sniffer is usually deployed to listen to all the traffic traveling across a particular network. This assumes a multiple access communication medium such as the one operating under the IEEE 802.3 CSMA/CD Ethernet standard so that the sniffer will be able to see the traffic for all entities on the network (even those not addressed to it). If a single access communication medium is used, the sniffer would have to be placed on a machine in front of the suspected victim to process the traffic before the victim receives it. Network interfaces normally ignore traffic addressed to other machines; however, this setting can be changed by placing the network interface into promiscuous mode. Network-based IDSs try to pick out signatures of various attacks in the data, raising alarms at suspicious behavior.

Due to their nature, there is some overlap in host-based and network-based attacks. For example, many host-based attacks actually occur from across the network. In addition, host-based systems can be effective in detecting outbound network-based attacks. An illustration of this overlap is shown in Figure 1.1. The detection methods for the different attacks also overlap. For example, if a login is made from across a network, the relevant data could be in both network data, and

host audit data.

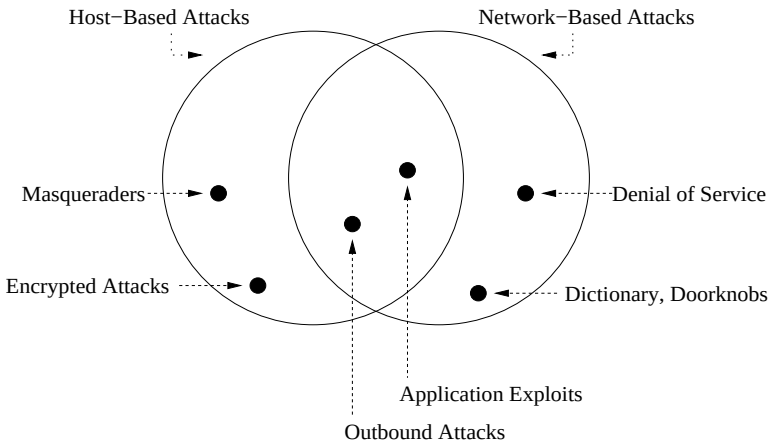


Figure 1.1: Illustration of overlap in host-based and network-based attacks

1.1.2.2 Intrusion Detection Approaches

There are two main approaches to detecting intrusions: anomaly detection and misuse detection.

- **Misuse Detection**

  The misuse detection approach attempts to model well-known attacks. Then, any behavior that matches the model is recognized as an attack. The major advantage of misuse detection is that well known attacks can easily be identified, so the reaction time can be reduced. The disadvantage however, is that no new attacks will be identified. This is the basic model that modern anti-virus software uses. The software can usually detect viruses it has seen before; however, you must constantly update the virus engine for detecting new viruses.

- **Anomaly Detection**

  Anomaly detection, by contrast, is the approach in which, first, a model of normal system or user behavior is created, and then, any behavior that deviates from the model is anomalous. The major advantage of this approach is that unknown or new attacks can be identified, because their pattern will be

a deviation from the model of expected behavior. However, these systems can cause many false alarms when valid changes in behavior occur.

Recent customer reviews have shown that the misuse detection tools can be difficult to configure [43]. This is especially the case in dynamic corporate environments. Small configuration mistakes can lead to large numbers of false alarms and missed attacks. Customers are pursuing anomaly based solutions claiming a drastic reduction in maintenance and updating time.

1.1.3 Congestion Control

Congestion control is an important part of any network management strategy. This is a function usually incorporated into the transport layer protocol of the OSI reference model [42]. The problem is that the routers which forward network packets in the right direction have a maximum output of μ, but may be able to receive packets at λ. If λ is close to μ, the random variations in both will force the router to store the excess packets received into a buffer. This increases the transfer time (delay) that a source experiences transmitting data to its destination. To take the problem one step further, these routers have a finite memory size for packet storage. Therefore, when this memory is full and temporary packet arrival rate exceeds their processing rate, the router must make a decision to drop some of the packets. In this case, the receiver will not receive all of the packets (and in a drastic case, no packets at all!). The transport layer protocols that attempt to address this point of unreliability, usually incorporate congestion control along with their efforts to assure successful data arrival.

1.1.4 Parameter Optimization

Many of the algorithms used to address the previous three issues have variable parameters which must be set to meaningful values to be effective. However, the interaction of one variable on another in the same or different algorithms can be difficult to understand. Currently, many of these variables are simply set by the administrator from his or her experience, or set to a default, often sub-optimal value.

A new initiative towards automating the administration of many complex systems from an Information Technology perspective, known as *Autonomic Computing*, has been created [79]. In this initiative, modeling of these complex systems helps to predict how a certain perturbation of parameters may affect the system as a whole. These models must be quick and efficient, however, if they can be used in real-time optimization techniques.

1.2 Overview

1.2.1 Problem Statement and Methodology

As computer networks grow in frequency of use, size, and degree of segmentation, network management applications must not only provide efficiency, non-intrusiveness and reliability, but also must be able to scale these characteristics over a wide range of network configurations. With the rising complexity of new network applications, scalability no longer means the ability of one machine to handle several requests at a time. Networking topologies have become specialized to the point that knowledge of the network is necessary for any network management application. Companies that have invested fortunes on networking infrastructure are not interested in adding additional infrastructure, but rather in fully utilizing their existing infrastructure design to perform at near-optimal levels given the specialized use of their network.

Our focus is to make network management as efficient as possible, providing a functional distribution of network managers scalable to the largest networks. We have created a distributed management framework to achieve this scalability along with several network managers, each contributing in their different fields of network management. In particular, we concentrate on the following strategies that we feel are necessary for an efficient, scalable network management application to implement.

- *Functionally distributed hierarchy*: The hierarchy should have the capacity to support the segmentation of the monitoring application. The segmentation ability is important for many different reasons, explained in the subsequent chapters. The most important among this reasons are:

- reducing use of network resources
- reducing load on individual machines
- decreasing response time to the client
- increasing data reliability,
- increasing data protection, assurance, and/or manipulation

Companies managing large networks may need to send management traffic across multiple Internet Service Providers (ISPs). However, many ISPs will not permit management traffic to travel across their networks [186]. In other cases, companies may want to protect their management or application traffic from in-route networks not under their control. In both cases, providing a functional distribution of the application would serve to relieve these constraints. Steps could be taken to transform, encrypt, or further manipulate the data, changing the communicated management data into normal (or encrypted) application traffic which is normally forwarded by in-route ISPs.

Some management protocols, like the Simple Network Management Protocol (SNMP), use an unreliable transport layer such as the User Datagram Protocol (UDP) for communication between manager and agent [130]. Network managers built on top of this protocol must take extra steps to not misrepresent the state of the network when management traffic is dropped [164]. For example, if the manager sends a request to the agent and receives no response, the manager would assume the device is down. However, the lack of response could be caused by a dropped management packet (either the request or the reply). Functionally distributing the application would allow for the use of different transport layer protocols in the most critical path between the manager and the agent.

It has also been shown that with certain management applications, such as Intrusion Detection systems, increased segmentation in networks has presented significant problems with access to needed data [116]. These systems rely on the ability for the network interface on a particular host to be placed in promiscuous mode and see all of the traffic flowing in the network, even

packets addressed to other hosts. However, with increasing segmentation of the network with routers, bridges, and switches, the size of the individual network that the promiscuous interface can see becomes smaller, causing it to miss segmented traffic. Functionally distributing the promiscuous processes into these segmented networks is an obvious way to use the same management applications in the segmented network.

Documented product comparisons of commercial IDSs have revealed problems with the scalability of almost all current systems [122]. Authors Joel Snyder and Rodney Thayer tested several commercial IDSs, only to find that in addition to accuracy and ease of use, they should evaluate IDSs on a third metric, uptime. The IDSs struggled to keep up with the traffic, crashing often and consuming all available memory and cpu resources. A functional distribution of the IDS could reduce the resource usage on particular hosts by segmenting the detection between different machines. This could also reduce the amount of traffic each host must evaluate by pushing the IDS deeper into the network.

- *Consolidation of requests*: If each manager is responsible for the collection of its own information, then the network would incur $N_{managers} * CollectionLoad$ where $N_{managers}$ is the number of network management clients requesting information and $CollectionLoad$ is the effect or load that a single collection effort has on the network. If we are able to remove the responsibility of gathering the statistics away from the client (to another part of the network) and consolidate similar requests, we could reduce the load to the troubled part of the network to $\frac{N_{managers}}{k} * CollectionLoad$, where k is a coefficient representing the number of clients requesting similar data. Networks with several management clients are common. A network management client can be any number of processes including but not limited to the following:

  - visualization clients - applications displaying real-time, graph like representations of current link or machine activity. One visualization client can actually display the current state of many components. In this case the visualization component would place multiple $CollectionLoad$'s on

the network.

  - proactive clients - applications actively monitoring various networking components to determine when a network fault or problem may occur before the negative effects of the problem are incurred. Simple proactive clients monitor certain values, constantly comparing them to static or dynamic thresholds. Once the threshold is exceeded, an alarm is raised and proactive actions are taken. However, other research based methods attempt to model the activity of the normal network and raise alarms when sufficient difference is seen between the modeled and current activity.
  - reactive clients - applications that are responding to a problem state. These clients are usually in an investigation mode attempting to find the cause of the current problem. These clients can be particularly dangerous because they usually add more stress to an area already identified as a problem area.

  Concern has been raised regarding the load that the monitoring application actually places on the network. Some management applications when used to monitor large networks place a significant amount of traffic on the very network they are trying to manage [149]. This not only wastes bandwidth that could be used for real network users, but if the network is near capacity, it could lead to delayed or dropped management packets leading to problems in data integrity and reliability discussed earlier.

- *Coordination of results*: By having the actual collection responsibility centralized, we provide an easy method for collaborative coordination of results. This refers to the ability of the data from multiple client requests to be aggregated, and manipulated in such a way that the collective statistics may add value beyond that provided by individual statistics.

- *Attention to appropriate time scale*: Some networking problems have a very small time window in which they can be addressed before the environment changes, rendering efforts that took longer than such a window ineffective [100]. Care must be taken to apply techniques that can converge on a solution while

the solution is still relevant to the current state of the network. To this end, we have used several machine learning techniques known to converge quickly (e.g., Multi-layered Perceptron-based neural networks and Self-Organizing Maps). We also employ the use of finite automata in several ways to build models to be followed in a timely fashion, adhering to the proper time scale.

- *Incorporating all available data into management structure*: The use of environment or hardware information while designing the management application can make the management application more effective. We have been able to incorporate topological network information to stress certain relationships in our machine learning methods.

Learning strategies have gained popularity as foundation of intelligent systems which can be applied to areas without much background information, and give useful results. Modeled after many naturally occurring phenomena, these strategies can many times provide solutions to very complicated problems containing many degrees of freedom. Because of this unique property, we embrace these strategies to develop solutions to complicated networking problems. As mentioned earlier we have worked in four areas of network management: monitoring and modeling, information security, congestion control, and parameter optimization. We will show that learning strategies can be particularly useful in all of these areas.

1.2.2 Thesis Organization

Network management can be accomplished in many ways. We review these methods and present our distributed network management framework in Chapter 2. There, we compare our framework with traditional network management architectures in a variety of network topologies and traffic situations through actual network experimentation. Our comparison allows us to analyze the scalability of each solution in terms of the load on the network and effect seen by any of the system's users. We also compare our framework and traditional methods in simulated environments too large for our networking lab to represent. To formalize the validity of our simulation methods, we have developed a way of segmenting the flows of the network

application into smaller, easily-simulated "separable" flows. This theory as well as all of our experimental and simulated results will also be presented in Chapter 2.

To further develop the idea of a functional distribution of networking applications, we have created many network managers, each contributing to their field in some way. We have developed two network-based intrusion detection managers described in Chapter 3. One of the managers uses neural networks and self-organizing maps, while the other uses Time Dependent Finite Automata. Chapter 4 explains our host-based intrusion detection method based on Probabilistic State Finite Automata. We have also developed a congestion control and arbitration manager using neural networks discussed in Chapter 5. We review our neural network modeler and network parameter optimization in Chapter 6, and finally conclude the thesis in Chapter 7. A glossary of acronyms used in this thesis has been included as a reference in Appendix E.

CHAPTER 2
DOORS: An Efficient, Scalable Distribution of Network Management

Rapid growth of computer network sizes and uses necessitate analysis of network application middleware in terms of its scalability as well as its performance. A fundamental requirement for efficient network management is non-interference with regular traffic during management data collection. This insures that there is no change in the dynamics of the network flows caused by management functions. The standard method of collecting network performance data is through polling. The manager simply polls the network devices that are of interest to obtain the required information. However, the additional network traffic and the delay incurred by polled responses returning to the manager can be expensive. If care is not taken to collect the monitoring data with small interference of the regular traffic, the data collection could distort the picture of the very network being probed. Also, current methods usually make no effort to insure that management traffic arrives to the manager in a timely fashion, and some make no effort to make sure the management data arrives at all. This could also affect the management application's view of the network.

Large scale management data collection presents additional problems for companies with subnetworks distributed geographically. Typically, such companies have no control over paths between their subnets. Many times, ISPs do not allow packets with well known management application port numbers to travel across their network for security reasons. In these cases, not only would the management application experience delays, but its management traffic may be discarded by in-transit networks. If management traffic is discarded, usually no information is returned to the management application, possibly causing the administrator to falsely believe the managed device is down.

In this chapter we first discuss many of the current industry standards and protocols for network management including some of the leading corporate products.

We then describe and analyze our distributed network management middleware based on agents that can be dispatched to locations where they can execute close to the managed nodes. We then go further to quantify the benefits of such a distribution by implementing real-time network managers using our distributed framework and measuring our framework against others in networks of different sizes, complexity, and utilization. Lastly, at the end of the chapter, we review other research efforts similar to ours.

2.1 Current Standards

Some of network management's oldest and most known protocols are SNMP and CMIP. SNMP uses a simple request and reply paradigm where an SNMP client contacts (through UDP) a networking device running an SNMP server with a request. The sever, in turn, replies to the client with the requested information. SNMP also contains very limited support for a push paradigm by using SNMP traps. SNMP, partially because of its simplicity, has gained a very large share of the network management market share. More information about SNMP can be found at [147]. CMIP [178] is much like SNMP but it is used for an OSI network instead of TCP/IP networks. Because OSI networks are not as popular as TCP/IP networks, standards were created to also use CMIP over TCP, called CMOT [178].

SNMP can be inefficient as a network management protocol when many SNMP request must be made. In many cases, when problems occur, management traffic in the problem area increases, worsening the problem. Problems with the protocol have been addressed by IETF and ISO through modifications of their management architecture. SNMPv2 introduced hierarchical decentralization through the concept of proxy agents [26]. The proxy agent simply acts as a client to a group of devices on behalf of a network management station. Another protocol derived from SNMP, RMON (Remote Monitoring), provides network administrators with an additional level of statistics kept by the RMON agent or probe and retrieved by an SNMP client [176]. The RMON specification defines a set of statistics and functions that can be exchanged between RMON-compliant console managers and network probes. This functionality of RMON to involve other RMON agents, provides network ad-

ministrators with a more comprehensive and global network-fault diagnosis, planning, and performance-tuning information. Although these decentralized features improve the state of SNMP, they do not provide the desired level of decentralization and functionality needed to cope with large networks [63].

New network management initiatives have been created by DMTF (Distributed Management Task Force), called CIM (Common Information Model) and WBEM (Web-Based Enterprise Management). CIM focuses on schema specifications for representing management data in a common format [45]. WBEM supports the transport of CIM using the XMLCIM Encoding Specification and CIM operations over HTTP [169].

The CTIT (Centre for Telematics and Information Technology) holds standard recommendations for TMN (the Telecommunications Management Network) [30, 131]. TMN suggests a conceptual separation between the network that is managed (the telecommunication network) and the network that transfers the management information (the Data Communication Network, DCN). Separating the management network from the telecommunication network prevents potential problems with fault management. In the case of a failure in the telecommunication network, the management network will still be able to access the failing components. This access gives TMN better fault management capabilities than management approaches like the traditional CMIP and SNMP. However, the separate management network requires additional equipment and transmission systems. Another limitation of TMN is that failures can also take place in the management network, making it necessary to manage the management network as well (meta-management).

Intel Corporation has proposed a system called COPS (Common Open Policy Service) that has also found its way into the standards committees [49]. COPS is a simple TCP client/server model for supporting policy control over QoS signaling protocols. The model makes no assumptions about policy server methods, but expects the server to return decisions to policy requests. The model is designed to be extensible so that other kinds of policy clients may be supported in the future.

Many corporate products have emerged, instituting their own set of features. Some of the most popular network management packages are HP's OpenView [77,

78], IBM/Tivoli's NetView [83], Intel's Policy-Based Network Management (PBNM) [50], and ProductivityNet's ActiveManage [132]. Most of these packages offer solutions based on special agents to be installed on the managed resource, and sophisticated management applications which display and report a variety of statistics to the administrator. Most of the corporate packages also have modules which interface some of the standard protocols and their corresponding agents which may already be installed on networking hardware (such as SNMP agents).

2.2 DOORS

To support the decentralization of network monitoring tools, we propose the DOORS system (Distributed Online Object Repositories) that facilitates scalable collection and manipulation of several forms of network data. The DOORS system manages and schedules client data requests at its repositories. The repositories then configure mobile agents to travel to a node very close to the managed device. Once the agent arrives at its destination, it polls the managed device, performs client requested procedures, and sends the result back to the repository to be forwarded to respective clients. The use of agents allows us to place more functionality into what the client perceives as the "request." A DOORS client may ask for various forms of direct network data, as well as any function $f(t, x_1, x_2, ..., x_n)$ of network data and time, where the argument x_i denotes data collected at discrete time i for $1 \leq i \leq n$. Typically, the function f is a statistical manipulation of network monitoring data. However, as we discuss later in the chapter, the function f may also involve more complex computations and management procedures. As the agent executes some functions at the remote location, the DOORS system effectively moves the computation closer to the data, instead of the traditional method of transporting all the data across the network to the client for computation. Moving the computation closer to the data drastically reduces the total bandwidth used by any tools which monitor large networks.

2.2.1 What is DOORS?

At the core, the DOORS system provides an efficient, fault-tolerant method for the acquisition, management, manipulation, aggregation, and caching of network data and objects [20]. We use mobile agents and a dynamic interface to support management and collection protocols such as CMIP and SNMP [147] as well as other more complex management functionalities. The DOORS system can provide network resource information for any number of demanding applications without leaving a heavy footprint on the network. Our goal is to make DOORS scalable to the largest existing networks with hundreds of thousands of nodes that cannot be managed using traditional, strictly hierarchical approaches.

Through DOORS, any client needing information about any node on the network posts requests for objects and data. The repository manages and responds to the client requests in a manner virtually transparent to the client. Through several mechanisms, DOORS takes steps to meet the temporal requirements of the requester. If the data in question is available in DOORS (for historical data), the object and its data are immediately provided to the requester. Otherwise, DOORS either communicates with an existing agent or launches an appropriate agent in order to provide the object to the requester.

For example, if the client wanted SNMP data from a router, the DOORS agent would move near the target router and periodically poll it, returning the collected data to the repository instead of having the client poll the router from a distance. This method roughly reduces the number data requests and replies by half and increases the scalability of data collection. DOORS agents use TCP to send the data back to its repositories in contrast to SNMP's UDP transport protocol. This choice makes our data collection more reliable as UDP packets may be dropped with no warning or repairing action. Scalability is increased by allowing requests outside the domain of a repository to be forwarded to an appropriate cooperating repository.

2.2.2 Impact of Data Location

Network management places complex requirements on the physical location of the network data. As eluded to earlier, three major factors of network management are performance, availability, and bandwidth usage. Performance, in this case, involves client queries and agent updates. For the clients and agents, repositories must be "nearby" in the sense of the physical layout of the network. This implies that the repository resides on the same subnet or at most a few hops away from clients that are assigned to it. However, it is unrealistic to expect a repository to reside in every subnet. One repository per some small set of physically close subnets should be sufficient. For performance reasons, the optimal location of the repository is the network region for which it is holding data. However, data for a network region should be available during periods when that region is unreachable. Therefore, the repository should be somewhere nearby without actually residing in the region.

Another important issue is bandwidth usage. One of the main goals of the DOORS system is to provide its services with an absolute minimum impact on the network. The repository itself assists in this goal by the functionality it passes to the client through the requesting language. The repository handles cases such as clients requesting a certain set of SNMP variables $S = \{v_1, v_2, ..., v_n\}$ every $t_{interval}$ seconds for a total duration of t_{total} seconds, in one command. This allows the system to use a push method of the client requesting data only once but having the data delivered to the client many times. If a system uses a standard network management protocol for the previous example, the client makes a separate request every $t_{interval}$ seconds for a duration of t_{total} seconds. This extra traffic may noticeably impact network performance, and make the results obtained for large networks useless. However, under DOORS, the client makes one request, an agent travels to a polling station, and the agent queries the machine (router) every $t_{interval}$ seconds for a duration of t_{total} seconds. The agent only sends the requested values back to the client without the client asking for them at every time interval. Applications such as network problem detection or prediction need large amounts of current data; therefore, minimizing the data collection impact on the network performance is of utmost importance.

2.2.3 DOORS Architecture

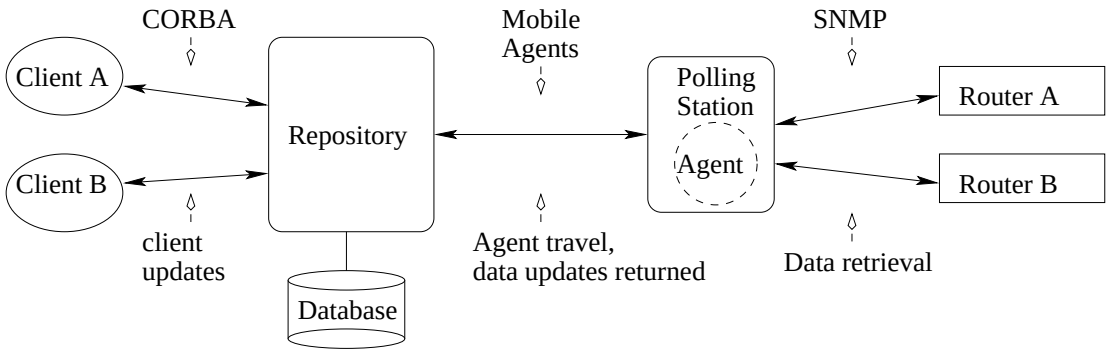


Figure 2.1: DOORS Architecture

The DOORS system uses several components to retrieve and process network data. A simple graphical representation of these components working together for a local data request (involving only one repository) is given in Figure 2.1. The purpose of each component will be given here, whereas the specifics, including what software packages we actually used, will be described in the next section.

2.2.3.1 Client Interface

The client interface is a simple program used to communicate with the repository. It may be a small stand-alone application or part of a larger application involving visualization, network problem avoidance, or other "upstream" application components. It only needs to formulate and send a request in a form recognizable to the repository. The clients in Figure 2.1, named "Client A" and "Client B", communicate with only one local repository.

2.2.3.2 Repository

The repository controls agents and coordinates requests from different clients as well as other repositories. Upon receiving a request, the repository determines if the request is for historical data. If the request is for historical data, the database is consulted for a response. If the request is for current data, the repository searches its existing agents to determine if there is already an agent collecting the requested data. If there is already an agent at the target router gathering relevant data, the

repository simply distributes the data being gathered to both clients. This provides tremendous savings when multiple clients need similar network data. In this way, the system functions much like a caching proxy, retrieving the popular data once but distributing it many times. Otherwise, of course, a new agent is sent to the appropriate location to gather the requested data. The chances of a large number of clients asking a particular repository for the same data may be low in today's network management suites; however, it is certainly reasonable to have more than one client need similar data, especially when tracking troubled locations in a network.

2.2.3.3 Mobile Agents

The mobile agents travel from the repository to a polling station to request the actual data from the destination object (typically a router). Agents reduce the network load by passing back only the data necessary for the client and the repository. This is especially useful when additional processing functionality is added to the agent. In such a case, the agent returns the result of some computation or manipulation of data, utilizing the paradigm of "taking the computations to the data, rather than the data to the computations." An example of such processing is stripping off the return value identification strings produced by SNMP and performing statistical calculations that the client normally performs and only sending the result.

Mobile agents also give us a great framework for updating and adding new functionality. The agent encapsulates protocols and procedures, so when a change needs to be made or functionality needs to be added, only the agent is impacted. Under any other implementation, each client and request receiving platform needs to be updated individually and consistently. Such an update is an expensive task to implement and manage on larger networks.

The agents execute asynchronously and autonomously, so once an agent is assigned to a job, the user is free to process other tasks. In addition, the agents are naturally heterogeneous. They are both hardware and operating system independent because they depend only on their executing environment.

Most agent packages, including DOORS, provide communication between agent

and sender using TCP as the underlying transport protocol. TCP provides reliability by re-sending packets which have not been acknowledged as received by the receiver. However, network management protocols like SNMP use UDP as their underlying transport, giving way to possible data losses in the process of communicating requests and results. The extra benefits of TCP are desirable but come at a significant cost in overhead; see Section 2.3.1.2 for an explanation of how we counteract these additional costs.

Perhaps the most important advantage of mobile agents is their robustness. Mobile agents can react dynamically to different conditions, which makes it easier to build a robust, fault-tolerant system. For example, if a host is shut down, all agents working on that machine may be given a warning and time to dispatch. This way the agent may collect the data from another machine or return to the issuer of the agent to explain exactly what happened. If a system uses conventional methods to collect data, either garbage is sent back, or an unpredictable error message is produced.

2.2.3.4 Polling Station

The polling station is a critical component, because it is difficult to send an agent to sit on the router itself to request data. Many routers use a custom operating system. Consequently, heterogeneity of platforms makes the code maintenance for the data collection difficult. In addition, running the collection programs directly on the routers may introduce unacceptable load on the routers. To avoid such an overload and for various security reasons, a group of routers may have a *polling station* allocated to it. This station can be viewed as a designated SNMP client for them. Our data collecting agent travels to the polling station to collect the data from the router at a close range, with minimum network traffic incurred by network segments between the repository and polling station. The polling station needs to run an agent server which can receive the agents and allow them to execute their tasks. Most agent servers, including our home-grown server, are lightweight applications, and do not need much CPU time or memory.

2.2.3.5 Storage System

A sophisticated storage system is not necessary for the basic functionality of the DOORS system. We use ObjectStore, a lightweight database system which stores objects for fast, easy retrieval [91]. ObjectStore holds historical and current data retrieved with the agents, as well as meta-data used to configure parts of the system. In addition to the storage system, the repository maintains a cache of the most recent data retrieved for quick matches with client requests [19].

2.2.4 Implementation Details

Building our prototype, we made several decisions regarding the tools used. Our primary concerns in selecting these tools were portability, extensibility, and ease of use.

2.2.4.1 CORBA

We have chosen to use the CORBA object model as the primary vehicle for communication and management of distributed network objects between our clients and repositories. Currently we are using the CORBA classes available in the Java 1.2.2 release along with the Java idl compiler. We chose this implementation of CORBA to increase the likelihood of our implementation being compatible with new JDK releases. Details about CORBA will not be provided here, since general information about CORBA is widely available (see, for example [13]).

2.2.4.2 Mobile Agents

Our network monitoring architecture requires the ability to carry out complex polling, data collection, and analysis on or close to the network component. The ability of the agent to analyze and gather statistics near the network component and return consolidated results is a significant source of improvement in performance. It has also been observed that installing, updating, and managing stationary agents on remote systems become prohibitively difficult in larger systems.

In order to address these problems, we use our own Mobile Agent Framework (MAF) [134] that enables easy development and deployment of custom agents. MAF takes over the responsibility for persistence, mobility, execution, communication

and security. It provides an architecture in which a custom agent can be built to carry out statistical computations and complex analysis of data collected from the network. The framework requires any such agent to implement a particular set of functions from an interface called RemoteExecution. The agent developer can define the autonomous behavior of the object and can control the actions of the agent by defining the corresponding methods.

Experimental use of some general purpose agent systems, such as the Aglets Agents [84], has shown that they could have a significant performance penalty because of the overhead involved. Our agent framework has been designed to minimize the agent transport and communication overhead. Deployment of agents in MAF is also easier than in most of the general purpose agent systems.

MAF provides a number of features and services to implement and deploy agents.

- Mobility

  The most important feature of the system is that it supports full mobility of agents. Once an agent that implements the required functions is determined to carry out safe execution, it can execute on any platform that runs the MAF agent server. As explained below, MAF also provides a standard set of libraries for communication, agent tracking, and directory lookup.

- Security

  The framework provides a sand-box model with configurable security at the agent server for the agent to execute. The agent is allowed to perform only a restricted set of actions so that it cannot cause harm to the system.

- File Management

  The system requires mobility of code, which is implemented by transfer and management of class files from one station to another. A complex system of versioning and class file caching is implemented for performance optimization.

- Service lookup

The system provides a dynamic lookup mechanism so that services can register themselves at a centralized directory and agents can lookup services from the directory. Hence agents can obtain information necessary for autonomous behavior dynamically.

The MAF architecture consists of three main parts:

1. Agent Server
2. Agent Sender
3. Lookup Directory

The agent server, which needs to be installed on the network polling stations for data collection and manipulation, provides a multi-threaded, secure management and mobility platform for the agents. The agent server registers services available at its host, with a lookup service called the SuperServer. The SuperServer handles lookup requests based on the services registered. In DOORS, the access to a router near the agent server (polling station) is treated as a service offered by that agent server. Lastly, the deployment is facilitated by a client helper called AgentSender.

In order to be deployed as an agent, a class needs to implement an interface called RemoteExecution. This interface forces the agent to define some functions such as:

• remoteProcess()	The function defining the actual work of the agent
• nextHost()	The function returning the agent's forwarding address
• processMessage()	The function processing messages sent to the agent

The system also provides storage of relevant agent addressing information and a set of powerful agent communication functions.

2.3 Laboratory Experiments and Results

As previously stated, we retrieve SNMP data from the routers that are targeted in client queries. Standard SNMP polls could be used to retrieve data from a distance instead of a more complex system such as the one that we have described

here. However, our slightly more complex system can provide fault-tolerance, ease of data management, and the ability to aggregate data and requests while placing the smallest possible footprint on the network. In this section, we compare the DOORS system to a traditional SNMP system in terms of their effect on the network and on the client application.

To determine the effect that each method had on the network, we monitored the bandwidth used by clients requesting data through the DOORS system with the bandwidth needed by the same requests executed using a standard SNMP client. We took the measurements over an isolated computer laboratory in the Computer Science Department at Rensselaer Polytechnic Institute. All of our tests were run for ten minutes using various polling intervals, each time requesting the following SNMP variables from one of our lab routers.

- 2.2.1.10.2 (*ifInOctets*)
- 2.2.1.16.2 (*ifOutOctets*)
- 4.3.0 (*ipInReceives*)
- 4.6.0 (*ipForwDatagrams*)
- 4.9.0 (*ipInDelivers*)
- 4.10.0 (*ipOutRequests*)

For more information about SNMP variables, see RFC 1213 [115]. The bandwidth usage metric we use, B, is defined in Equation 2.1. All counter variables used in Equation 2.1 are obtained using a traditional SNMP client running concurrently with the management application being evaluated.

$$B = \frac{(ifInOctets_{last} - ifInOctets_{first}) + (ifOutOctets_{last} - ifOutOctets_{first})}{t_{last} - t_{first}} \tag{2.1}$$

t_{first}	timestamp of the first SNMP variable's arrival at the client
t_{last}	timestamp of the last SNMP variable's arrival at the client
$ifInOctets_{first}$	count of the number of octets that have traveled into the router's s1 interface at timestamp t_{first}
$ifOutOctets_{first}$	count of the number of octets that have traveled out of router's s1 interface at timestamp t_{first}
$ifInOctets_{last}$	count of the number of octets that have traveled into router's s1 interface at timestamp t_{last}
$ifOutOctets_{last}$	count of the number of octets that have traveled out of router's s1 interface at timestamp t_{last}

To measure how each management application impacts the underlying client, we look at client based statistics related to when the client actually received the requested data. In particular, we ran a statistical analysis of the data received by the client to examine the variance in the times between successive returns of information, or inter-polling time. We compare this variance by calculating the standard deviation of the inter-polling time of the clients. We use two standard deviation statistics, average-based standard deviation and actual-based standard deviation (illustrated in Equation 2.2 and Equation 2.3 respectively). The average-based standard deviation evaluates the standard deviation through computation of each interval's difference from the sample's average interval (also known as sample standard deviation). This will measure the regularity and smoothness of the inter-polling interval (the lower this deviation the more consistent the inter-polling interval). The actual-based standard deviation involves computation of each interval's difference from the polling interval requested by the client. This deviation statistic measures how close the intervals are to the actual interval requested by the client. In our cases, the data integrity was perfect, meaning that all polled values were returned successfully.

$$sd_{average} = \sqrt{\frac{\sum_{i=1}^{n}\left(Im_i - \frac{\sum_{j=1}^{n} Im_j}{n}\right)^2}{n-1}} \tag{2.2}$$

$$sd_{actual} = \sqrt{\frac{\sum_{i=1}^{n} (Im_i - Ia)^2}{n}} \quad (2.3)$$

Im_i	interval measured between successive returns of data (if a timeout occurs, that particular interval is ignored).
Ia	interval requested by the client.
n	the number of successful data retrievals.

2.3.1 Isolated Small Network Tests

Our small network testbed consisted of six machines running FreeBSD and three Cisco 2500 series routers. The topology is shown in Figure 2.2. The two Ethernet networks are each composed of three machines connected by a 10MB hub. Connecting these two subnets are three routers, which are connected through serial links and run the Routing Information Protocol (RIP). The RIP protocol requires the routers to send updates every thirty seconds advertising how many hops it will take them to reach various networks [82]. This is relatively light traffic because our topology consists of only three routers. In all measurements, we collect the SNMP data from the Ethernet interface of router r3. In an effort to make this network give us the effect of a larger network, the clock-rates of the router interfaces were reduced to that of a 5600 baud modem. A ping test from one side of the network to the other (n7 to n10) took an average of 54.9ms.

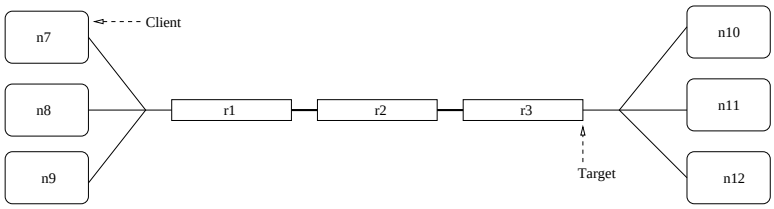


Figure 2.2: Topology of the Testbed when simple SNMP requests are run

We compare the amount of bandwidth used when single and multiple clients are present by calculating the bandwidth usage metric, B, from SNMP variables gathered at the serial1 interface of router r2 (which is in the middle of the topology shown in Figure 2.2). When standard SNMP methods are used, our topology is that

of Figure 2.2, where the client is running on n7 and we poll the Ethernet interface of router r3 (the target router).

The DOORS system architecture with the same topology is shown in Figure 2.3, with the client on n7, the naming server and repository both on n8, the super server on n9, and the polling station across the network on n10.

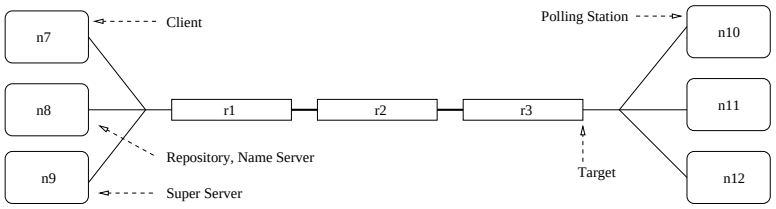


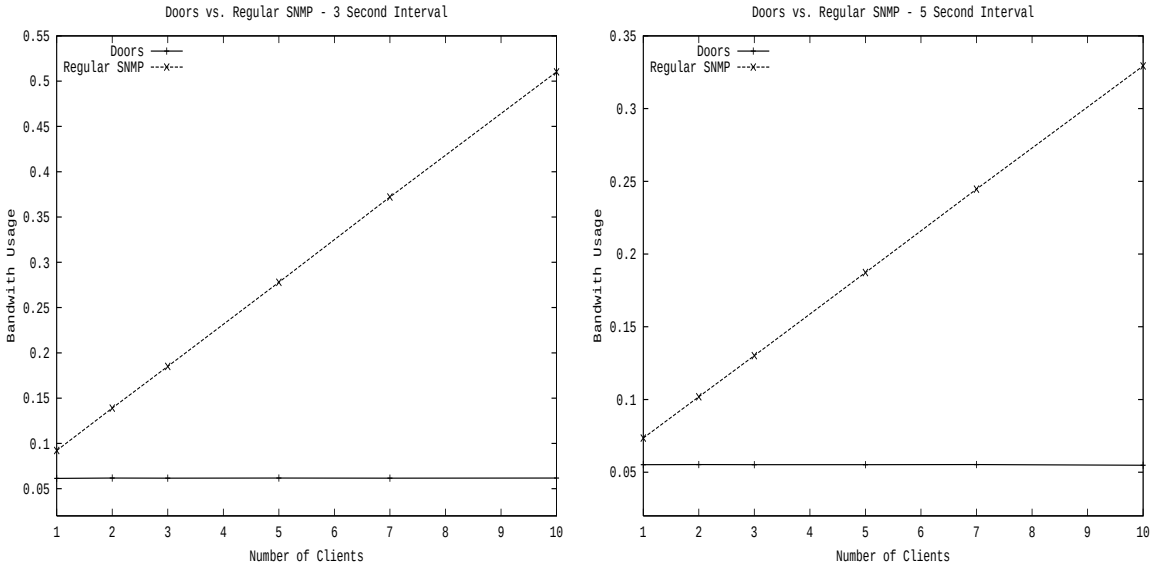
Figure 2.3: Topology of the Testbed for DOORS

2.3.1.1 Traditional Data Collection Case

In this case, each system is to collect the six SNMP variables previously listed at regular intervals for 10 minutes. These tests were run according to the topologies shown in Figures 2.2 and 2.3.

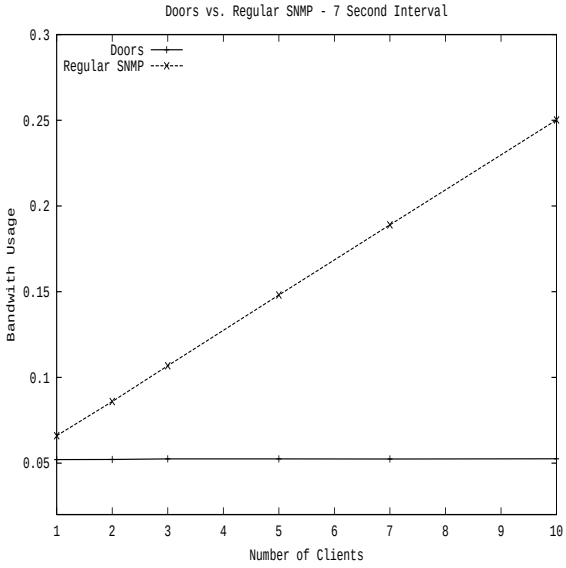
The graphs in Figure 2.4 show the bandwidth usage, calculated from Equation 2.1, incurred through data collection at several different intervals with the DOORS system versus bandwidth used by a traditional SNMP system. The DOORS system achieves better performance regardless of the number of clients used. Moreover, the bandwidth used by the DOORS system remains effectively constant regardless of the number of clients, because in all cases only one message per data collection interval is sent back to the repository to be distributed to any number of clients. However, the standard SNMP method must send a request across the network every polling interval for every client. Therefore, as the number of clients grows in data collection under the standard SNMP, so does the collection bandwidth used.

Our goal is two fold, we want to provide an effective way of data collection while putting a minimum strain on the network, but at the same time, the clients need to receive the data in a timely fashion. We have already shown how DOORS meets the first goal. The graphs describing the client-based statistics can be found

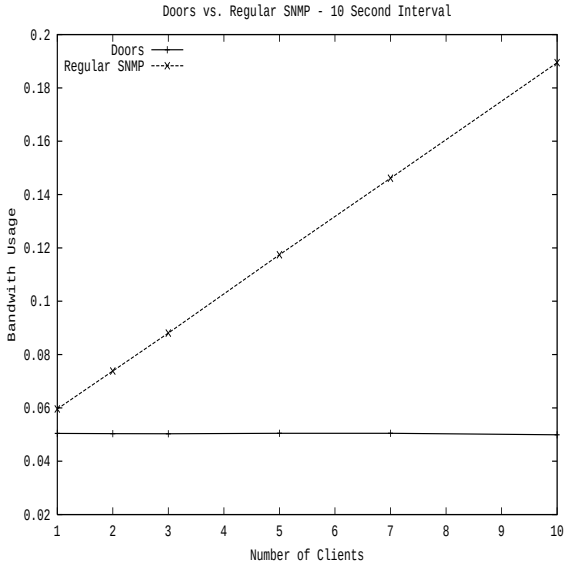


(a) 3 sec. interval

(b) 5 sec. interval



(c) 7 sec. interval



(d) 10 sec. interval

Figure 2.4: 3, 5, 7, and 10 second interval bandwidth usage plots

in Figures 2.5-2.6. It adds little value to display these statistics for each case we ran; therefore, Figures 2.5-2.6 only show the results for intervals used in the 3 client scenario.

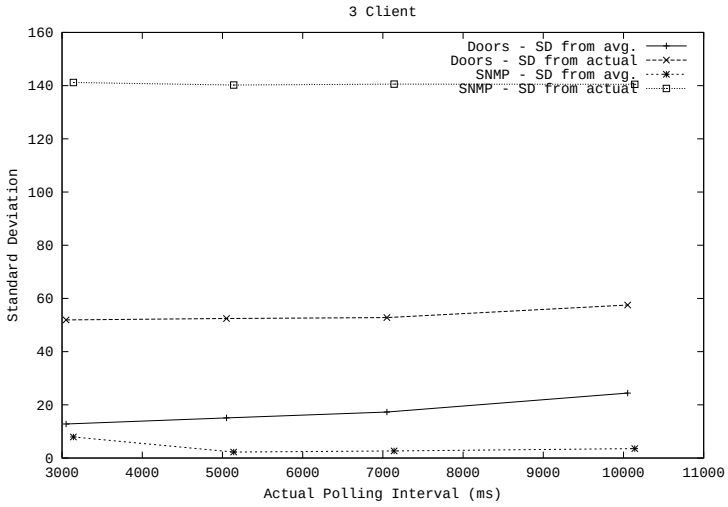
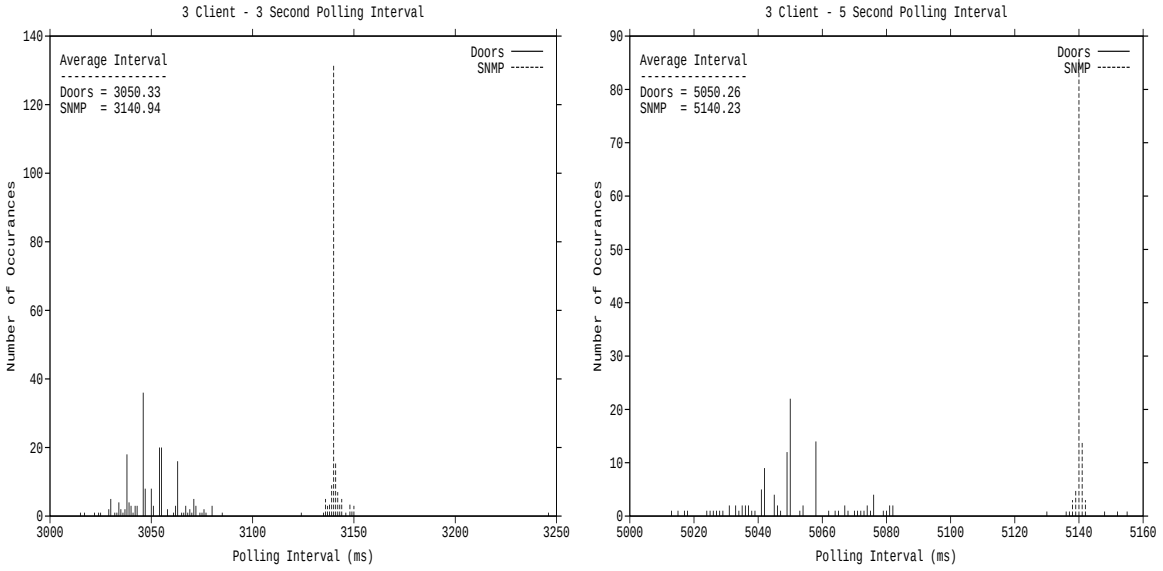


Figure 2.5: 3 client isolated network case, standard deviation plot

As illustrated in Figure 2.5, the normal SNMP client has a low average-based variance because, using UDP as a transport protocol, its inter-poll time is based only on the network latency and load which are relatively constant in the isolated network.

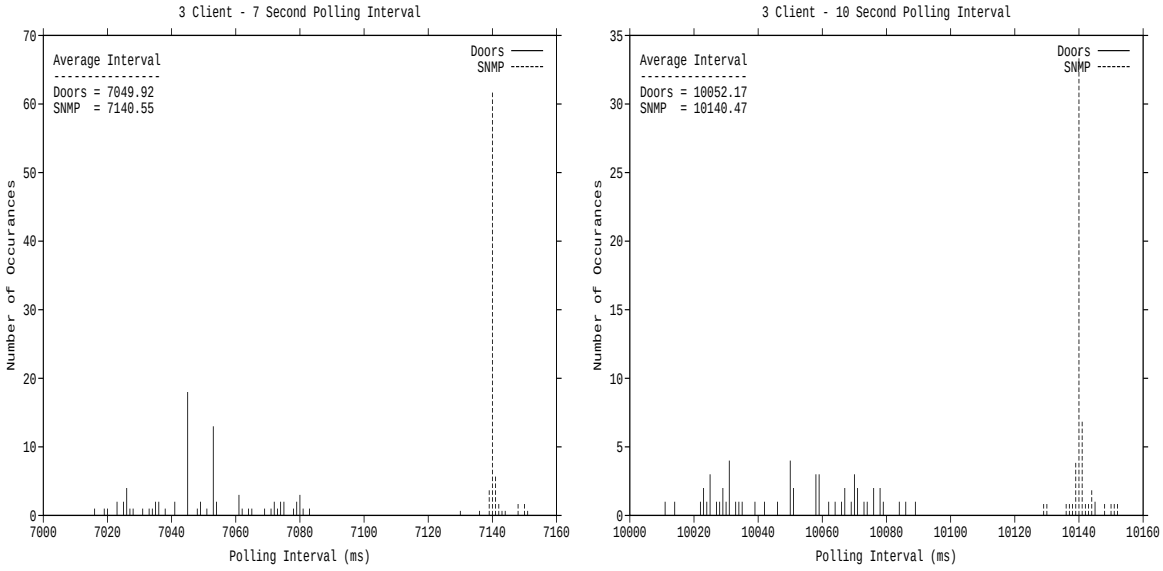
Because our agents use TCP to send data back to the client, we see a difference in the variance of the polling interval perceived at the client. TCP uses many different algorithms, such as control loop based flow control, slow start, congestion avoidance, and its windowing mechanism. The collective use of these algorithms can cause delays and oscillations in TCP delivery times compared to that of UDP [55, 92, 152]. However, because DOORS sends the configured agent close to the router, DOORS does not suffer the delays of traversing the network for the request portion of the request-reply paradigm that the normal SNMP client uses. Our agent also has an open connection with the repository for long-term message passing. These facts allow DOORS to have a fraction of the latency time the normal SNMP client has, and thus reduces its actual-based standard deviation.

Figure 2.6 shows histogram plots from the different intervals of the three client



(a) 3 sec. interval

(b) 5 sec. interval



(c) 7 sec. interval

(d) 10 sec. interval

Figure 2.6: 3 client isolated network case, (3, 5, 7, and 10 second interval histograms)

case. The dashed lines in these graphs represent the number of times the SNMP clients experienced the corresponding inter-polling interval. The solid lines show the same for the clients of the DOORS system. The average polling interval for each method is also displayed in the top left corner of the graph. The graphs show that the SNMP inter-polling intervals almost form one straight vertical line because of their consistency. The DOORS system inter-polling intervals are a bit more distributed. As seen in Figure 2.5, even though the standard deviation of DOORS about the average is much higher than that of the SNMP client, the deviation from the requested interval of the DOORS client is much lower than that of the SNMP client. This basically means that DOORS may be a bit more variable in the exact times it returns the data, but it will almost always return it faster than the SNMP client. The other side of this comment is also true. The SNMP client will almost always take longer to return the data, but it will return it at a more consistent interval. Both of these conclusions are confirmed in the histogram plots from Figure 2.6. Once again, in this particular set of test runs, both methods always return all of the data requested.

2.3.1.2 Analysis of Measurements

It is obvious that the DOORS system would use far less bandwidth than a traditional SNMP system when multiple clients requesting similar data are used. The most interesting results, however, came from the single client case. The DOORS system uses TCP for agent–to–repository communication while the standard SNMP method uses UDP for all communication. However, SNMP would need both a request and a reply (a pull method) while DOORS simply sends the update data (a push method). Because of the differences in the underlying protocols and the methods used to get data from the two systems, we will compare the two systems on a transactional basis. In this context a transaction is simply the client receiving a new instance of data at the polling interval requested. In this section we compare the standard SNMP transaction with the DOORS transaction by examining packet sizes of messages sent. All packet measurements were obtained by using tcpdump on a machine in the same network with an additional four bytes added to reflect the

checksum.

The Standard SNMP Transaction

SNMP usually works on a request-reply basis. The client sends an SNMP request and receives an SNMP reply, both of which are usually small enough to fit in one packet. This arrangement results in two packets: $Snmp_{request}$ and $Snmp_{reply}$. The same message format, containing SNMP headers followed by name-value pairs, is used for both the request and reply (see Figure 2.7(c)). The only difference between the two messages is that value fields are not populated with the actual values in the request packet. Due to SNMP's use of ASN.1 and the BER (Basic Encoding Rules) encoding standard, the difference in bytes between these two depends on the type of data sent and sometimes on the data values themselves [147, 28, 29]. The sizes for the SNMP transaction packets used in our experiments can be found in Equations 2.4 and 2.5. These values include Ethernet, IP, UDP, and SNMP header sizes (see Figure 2.7).

$$Snmp_{request} = 161\ bytes \tag{2.4}$$

$$Snmp_{reply} = 175\ bytes \tag{2.5}$$

The DOORS Transaction

As previously discussed, the DOORS system returns data using TCP for transport layer communication. Part of TCP's overhead is caused by its three-way handshake [151]. However, the DOORS agent maintains the connection with the repository over the entire collection period, so the single connection handshake overhead can be amortized over the life of the connection. Therefore, the difference between the bandwidth used by the two methods must be found in the data messages sent back from the agent. In TCP, data must be acknowledged with a special ACK packet. In our case this packet is a simple acknowledgment, but its size must be 60 bytes (the minimum Ethernet frame size). We have no need to send a request, so the only packet we send is the new data packet, holding the data and some syn-

chronization content for the agent system. The sizes for the DOORS transaction packets can be found in Equations 2.6 and 2.7. These figures include Ethernet, IP, and TCP header sizes (see Figure 2.7).

$$DOORS_{data} = 122\ bytes \tag{2.6}$$

$$DOORS_{TCP-ack} = 64\ bytes \tag{2.7}$$

As seen in Equations 2.6 and 2.7, the data packet from the DOORS application is much smaller than that of the $Snmp_{request}$ or the $Snmp_{reply}$. This is because the DOORS agent strips the object identifiers (SNMP labels) from the reply before sending the values back. Using BER encoding, SNMP can easily use 7-9 bytes simply to send an object identifier. Our agent sends the values back in a simple text format.

Comparisons

In summary, SNMP and DOORS transactions can be evaluated as follows:

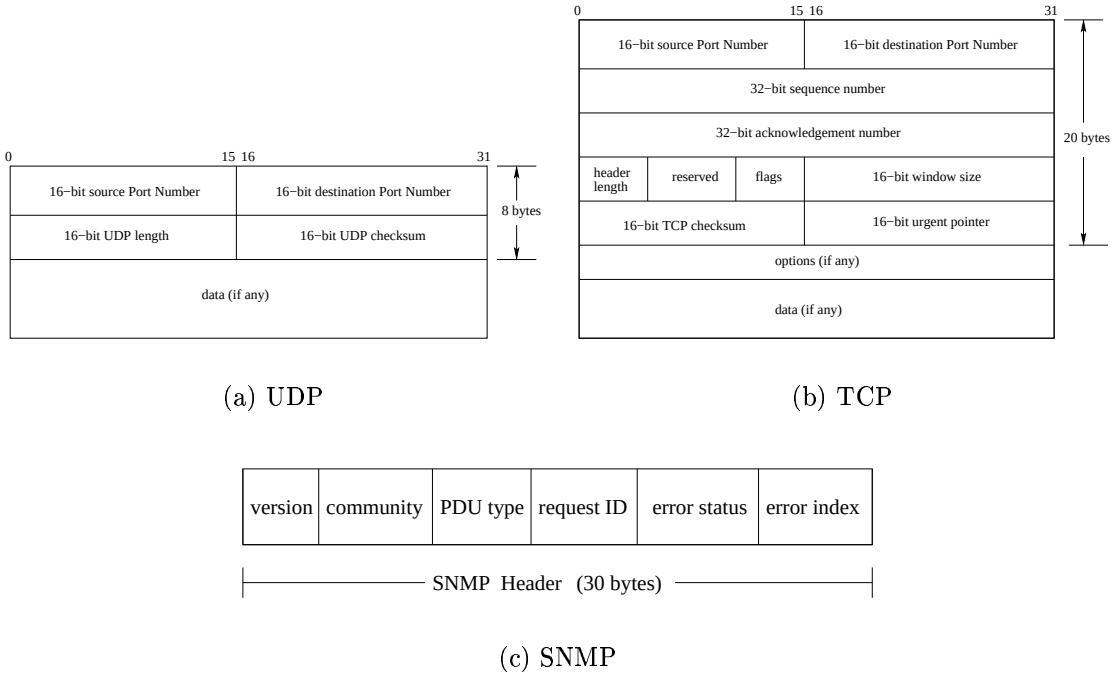
$$Transaction_{snmp} = Snmp_{request} + Snmp_{reply} = 336\ bytes \tag{2.8}$$

$$Transaction_{door} = DOORS_{data} + DOORS_{TCP-ack} = 186\ bytes \tag{2.9}$$

As Equations 2.8 and 2.9 indicate, DOORS is an efficient solution to data collection which lightens the footprint of any network management application. When the number of clients grow, the relative bandwidth used by DOORS remains constant while the standard SNMP method increases bandwidth usage at least linearly with the number of clients.

2.3.1.3 Preprocessing Case

Many research efforts use SNMP to gather vital network statistics and determine trends in the traffic traveling across the network. DOORS can play an integral roll in these applications by reliably collecting the needed data close to the device in question, and doing part or all of the necessary calculations in the agent, sending only the results of manipulations of traffic data back to the requesting client. To evaluate



(a) UDP

(b) TCP

(c) SNMP

Figure 2.7: Protocol Headers

the contribution of DOORS in such a situation, we use DOORS as a component of the Network Problem Forecasting solution developed by Thottan and Ji [168, 167]. These authors detect changes in traffic patterns using a sequential Generalized Likelihood Ratio (GLR) test. They first gather data using standard SNMP polling to obtain values of six MIB variables forming six time series. This data is then split into windows of ten time intervals to create piecewise stationary Auto-Regressive models. Using coefficients computed from each window, a sequential hypothesis test was performed using the Generalized Likelihood Ratio to determine the extent of statistical deviation between two adjacent time windows. Once changes are detected using the GLR, the authors correlated the different alarms through some specialized correlation techniques.

Using these methods, the authors were particularly successful in detecting when the network file system (NFS) stopped responding, resulting in a network fault that was easy to confirm using the system logs. The prediction horizon ranged anywhere from an hour before NFS crashed to 15 minutes after the crash.

We implemented the windowing and auto-regressive calculations in the traveling agent, sending only the likelihood values back to the client for filtering and comparisons. Thottan and Ji had to conduct normal SNMP polls to retrieve the data necessary for the calculations. However, our agent goes very close to the router in question and collects windows of polls, computes the likelihood values using auto-regressive techniques, and finally only sends back the likelihood values to the client. This division of the algorithm saves the cost of $N-1$ polls across the network per time window, where N is the number of intervals used in each window, because only one set of statistics is sent back per window versus a set of data for every poll. Hence, this solution reduces the bandwidth used by the factor of at least $N-1$.

The preprocessing tests were conducted in the same isolated Computer Laboratory of the Computer Science department at RPI. The network topology is the same as the previous tests (shown in Figures 2.2 and 2.3). The DOORS agent was responsible for computing the window-based likelihood functions as defined by Thottan and Ji [168, 167]. Specifically, the agent performed the following actions and calculations:

- Travel to the polling station and begin polling the managed device at regular intervals.
- Group the poll responses in windows of N responses each, forming a piecewise time series. An illustration of the windowing scheme can be found in Figure 2.8 where two adjacent time windows R(t) and S(t) of length N ($N = 10$) are given for a particular MIB variable.

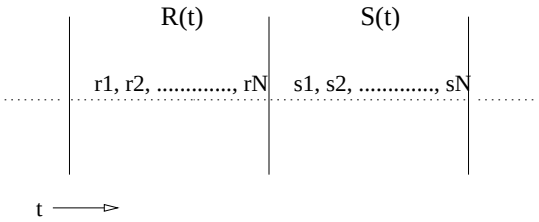


Figure 2.8: Illustration of the windowing of SNMP values

- Once the time windows have been established, the likelihood value of the residual time series can be obtained by the formula in Equation 2.10 (please see [168, 167] for formula derivation).

$$l_R = \left(\frac{1}{\sqrt{2\pi\sigma_R^2}} \right)^{\acute{N}_R} exp\left(\frac{-\acute{N}_R \hat{\sigma}_R^2}{2\sigma_R^2} \right) \tag{2.10}$$

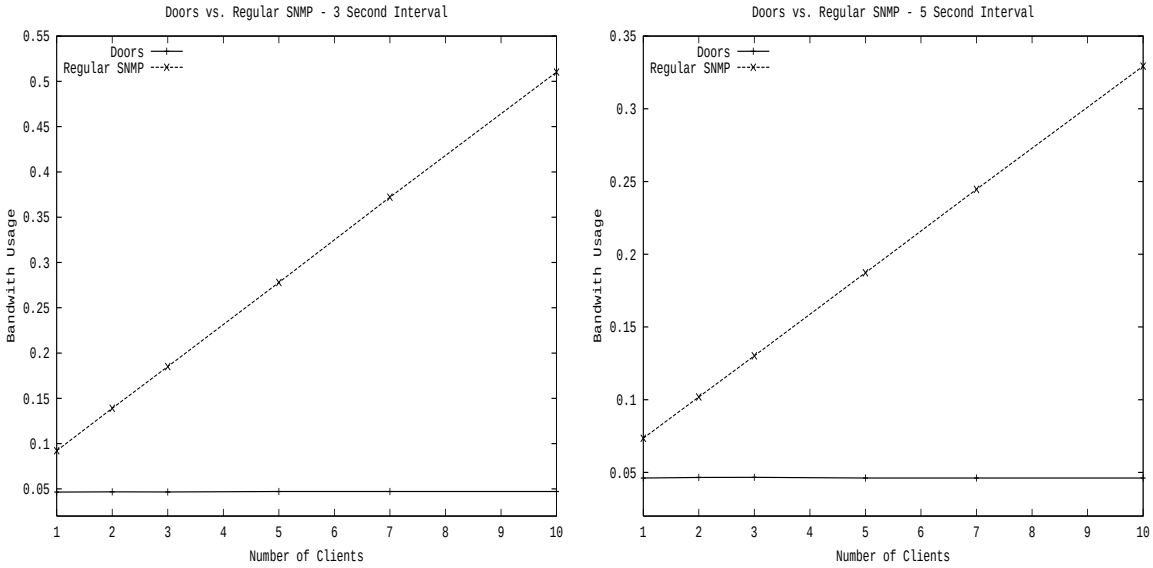
σ_R^2	=	variance of the residual in $R(t)$
$\acute{N}_R$	=	$N_R - p$ where:
		•N_R is the number of values taken in $R(t)$ ($N_R = 10$ in our case)
		•p is the order of the Auto-Regressive process used ($p = 1$ in our case)
$\hat{\sigma}_R^2$	=	covariance estimate of σ_R^2 (See [44])

- Return the value calculated (l_R) for each window of intervals instead of returning the polled values for each interval saving bandwidth usage and enabling for faster processing of the data.

The graphs presented in Figure 2.9 show that the auto-regressive agent uses very little bandwidth. In this case, the bandwidth usage metric, B, caused by the background traffic of the routers using the Routing Information Protocol is 0.038. Our measured bandwidth usage metrics cannot go below that point, but the preprocessing agent gets close.

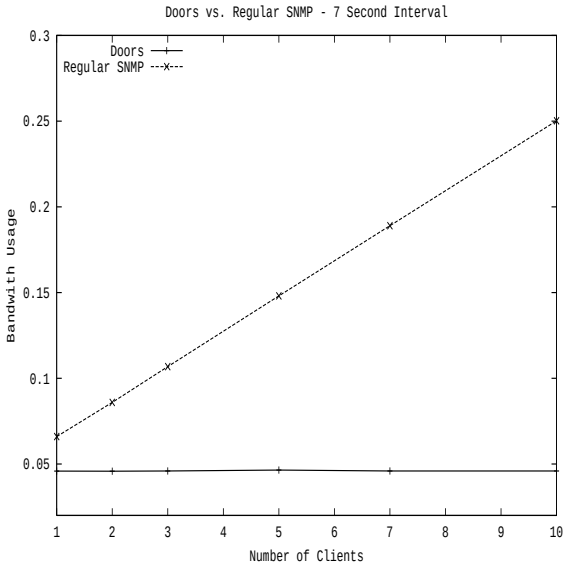
It is difficult to compare the standard deviations of DOORS with the auto-regression code to that of normal SNMP because now the polling interval perceived by the client is of a different magnitude. The client should only get a message once every 10 time intervals, while the normal SNMP client gets 10 times that many messages. However, to be consistent we ran the same comparisons. Because the intervals are now different magnitudes (shown in Figure 2.10), their standard deviations are also different magnitudes (shown in Figure 2.11).

The histogram plots (Figure 2.10) show the differences in the number of messages sent using the DOORS autoregressive agent and the number sent using the normal SNMP client. The very short lines represent the distribution of DOORS

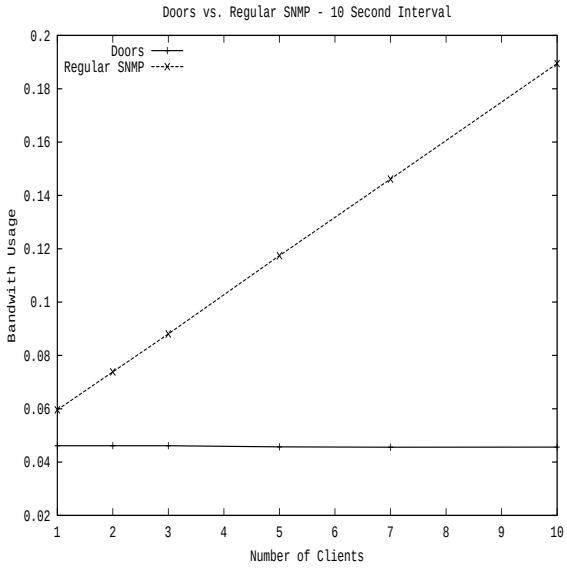


(a) 3 sec. interval

(b) 5 sec. interval

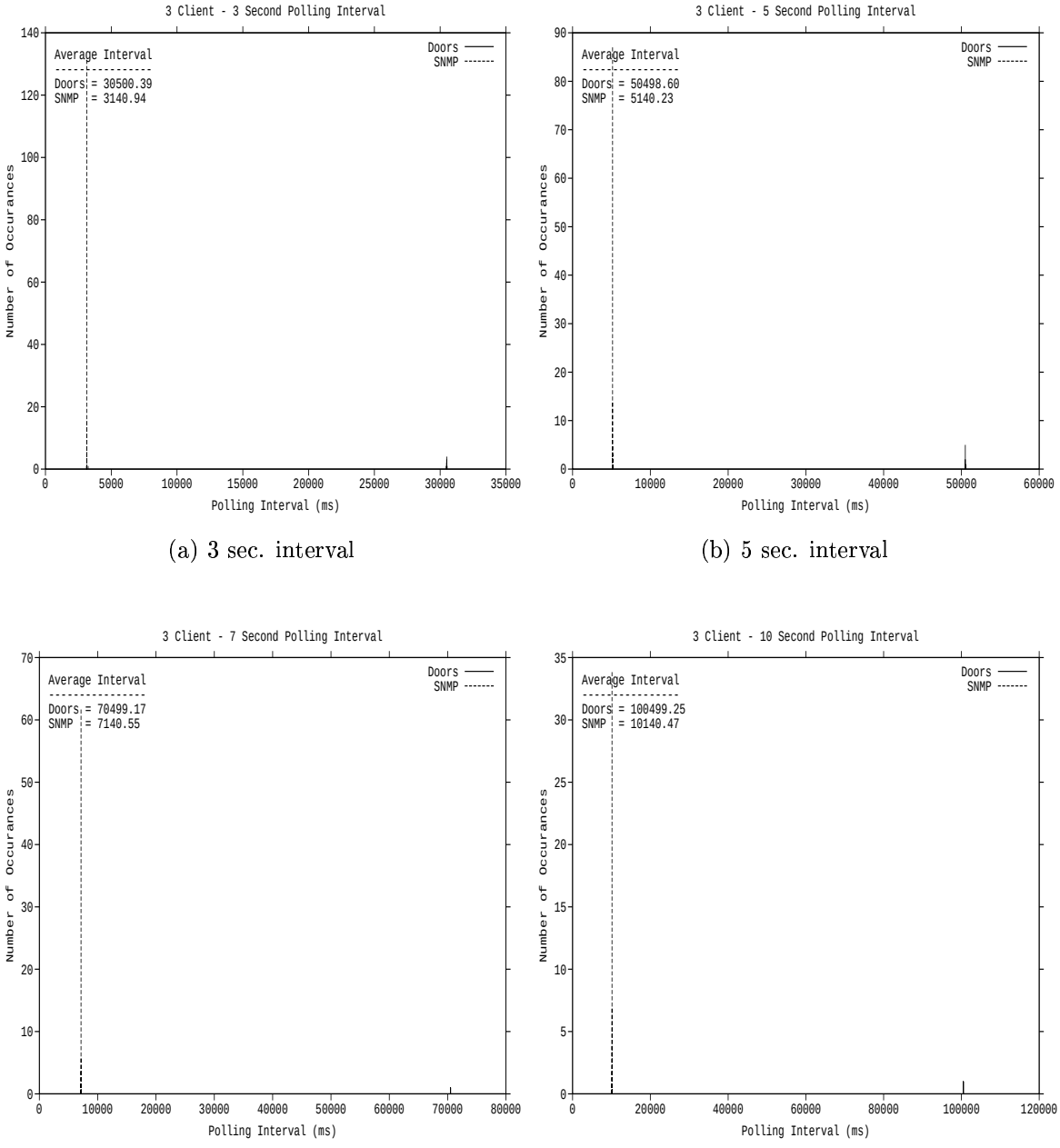


(c) 7 sec. interval



(d) 10 sec. interval

Figure 2.9: 3 client preprocessing isolated network case (3, 5, 7, and 10 second interval bandwidth usage plots)



(a) 3 sec. interval

(b) 5 sec. interval

(c) 7 sec. interval

(d) 10 sec. interval

Figure 2.10: 3 client preprocessing isolated network case (3, 5, 7, and 10 second interval histograms)

intervals. However, because the DOORS client is only sending one response for each window, the actual DOORS polling intervals are about 10 times larger than those of the SNMP polling intervals. Considering this difference in scale, the statistics look the way we would expect them to look.

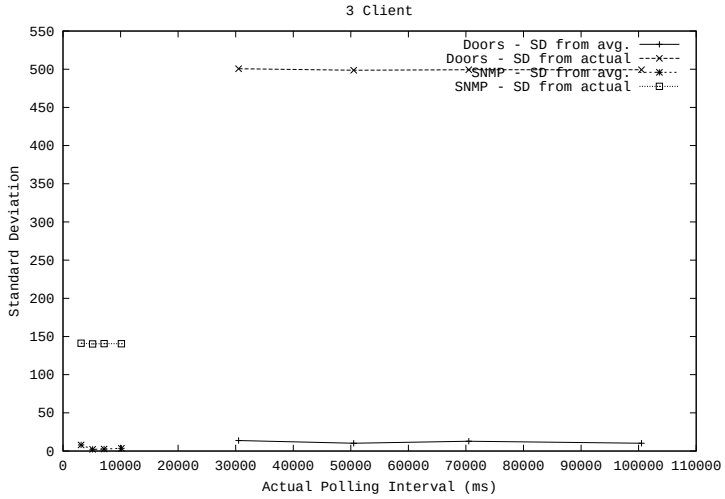


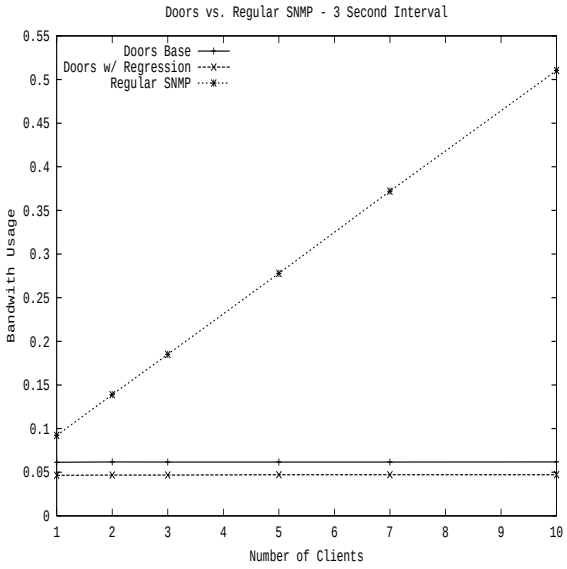
Figure 2.11: 3 client preprocessing isolated network case, standard deviation plot

2.3.1.4 Comparisons of All Methods

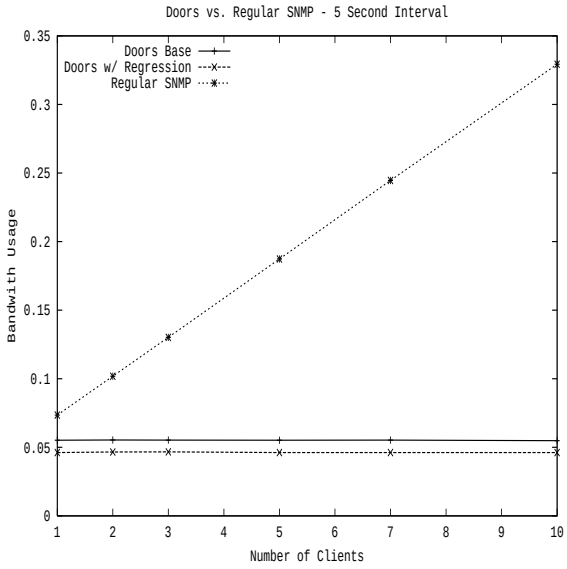
To show the differences in bandwidth usage across the many methods including the placement of part of the algorithm in the agent, we present the following graphical comparisons located in Figure 2.12. The graphs show the SNMP clients using considerably more bandwidth than the DOORS client. They then go further to show the additional savings to be had by placing part of the algorithm in the agent.

2.3.2 Internet Case

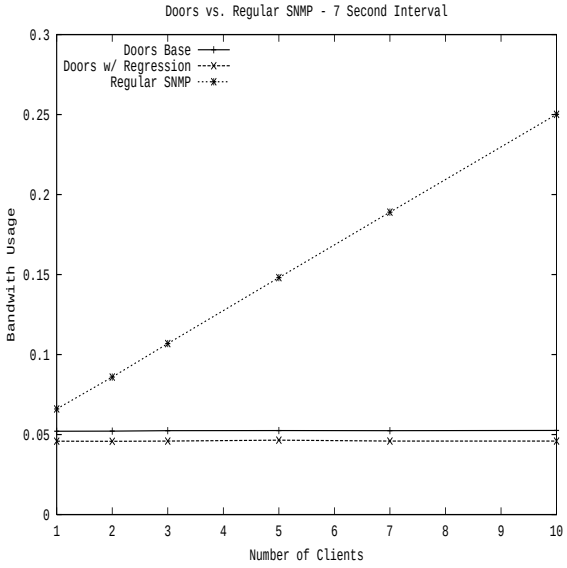
Many of the test details had to be changed for the case where the DOORS system is run across the Internet. In our Internet case, we set up the client, name server, and repository in our lab. However, the polling station was in a home network about fifteen hops away (See Appendix B for a traceroute between our client and



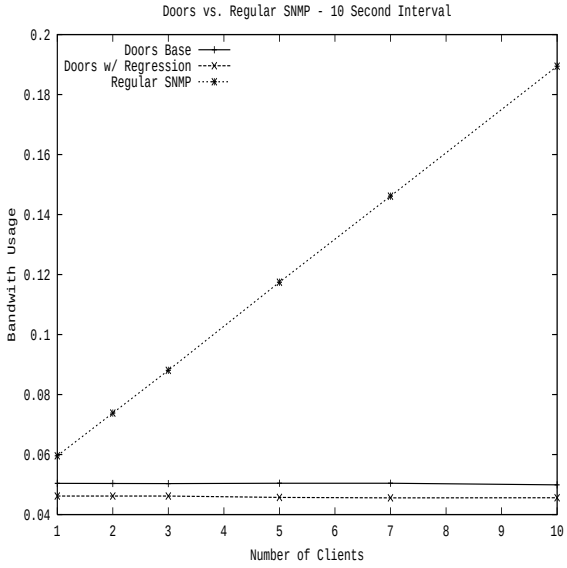
(a) 3 sec. interval



(b) 5 sec. interval



(c) 7 sec. interval



(d) 10 sec. interval

Figure 2.12: Joint isolated network plots of 3, 5, 7, and 10 second intervals

the home network). The topology of the Internet case can be seen in Figure 2.13, along with a short description of all non-host equipment.

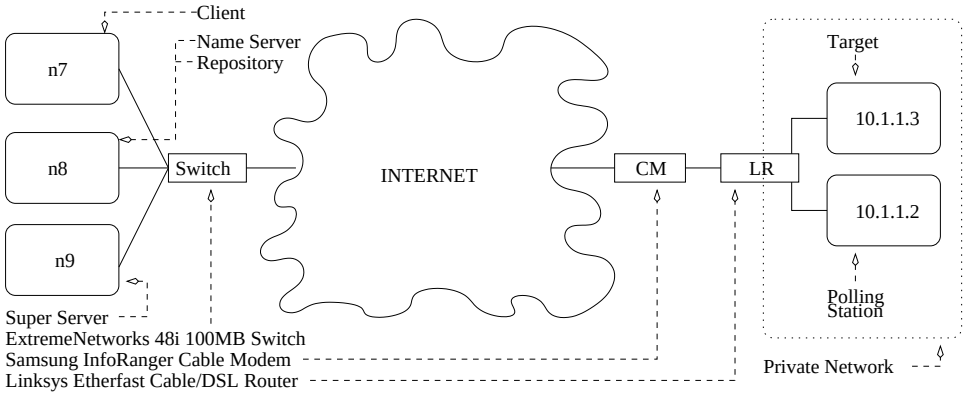


Figure 2.13: Topology of the Internet Case

The hosts on the home network were two Linux boxes. The polling station was a 450MHz Pentium II, and the target was a 300MHz Pentium I. The home network subscribed to a local cable modem broadband service. To connect multiple computers to this network, the users installed a Linksys Etherfast Cable/DSL router which acts as a NAT address translator, giving fake IP addresses to the machines behind it while allowing them to be connected to the Internet. This presented a significant problem because the only real IP address is given to the router, not to our polling station (which our agent needs) or our target (which the normal SNMP polls need). To circumvent these problems, a special port forwarding rule was created in the router to forward all requests to port 161 to the target machine. A second rule was created to send all other port requests to the polling station machine (placed the polling station in the demilitarized zone or DMZ). This would enable requests to be sent to the real IP address of the router and be subsequently forwarded to the correct machine. This allowed our agents to travel to the router and get forwarded to the polling station.

The lack of control of intermediate networks proved to be a second problem in this case. The cable provider did not allow SNMP requests to enter its network from an outside address. As mentioned before, DOORS is a natural solution to this type of problem. It simply sends its agent to the home network which we control,

allowing us to poll hassle-free. To provide a comparison to the normal SNMP client we evaded the cable provider's SNMP restriction by running an additional SNMP daemon on a different port and sending the normal SNMP client request to that port (fooling the detection system which is looking for packets heading to the default SNMP port, 161). Therefore, a second SNMP daemon was started on the target on port 9050, and corresponding port forwarding rules were set up in the home router.

Because these tests were run over the open Internet, outside traffic may significantly impact any results we attempt to monitor. For this reason, bandwidth comparisons would be useless as we certainly could not guarantee that the bandwidth was consumed by our applications. We can however, monitor the effect on the client (even though it too can be impacted by outside traffic). To this end we calculated the standard deviations of the inter-poll time perceived by the client in Figures (2.14-2.15). Due to the very dynamic changing patterns in Internet traffic, we interlaced our normal SNMP tests with those of the DOORS application to minimize the difference in Internet state while running corresponding tests. Although this gives us no guarantee that the state of the Internet will be the same, it is the best that could practically be done.

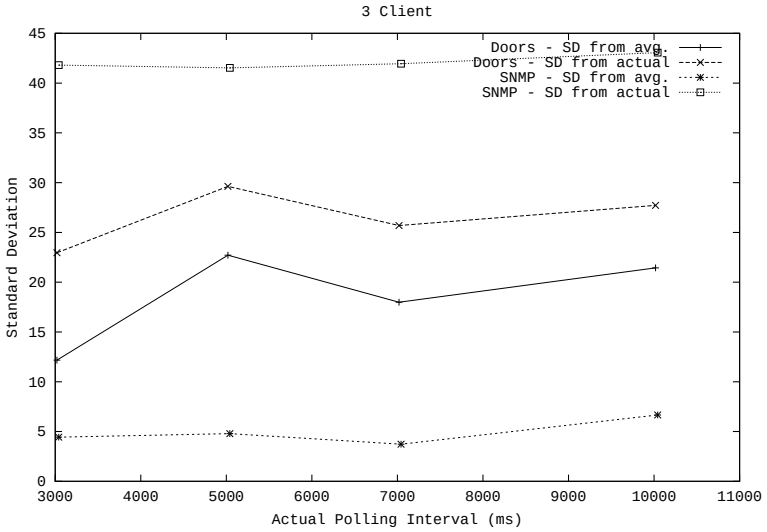
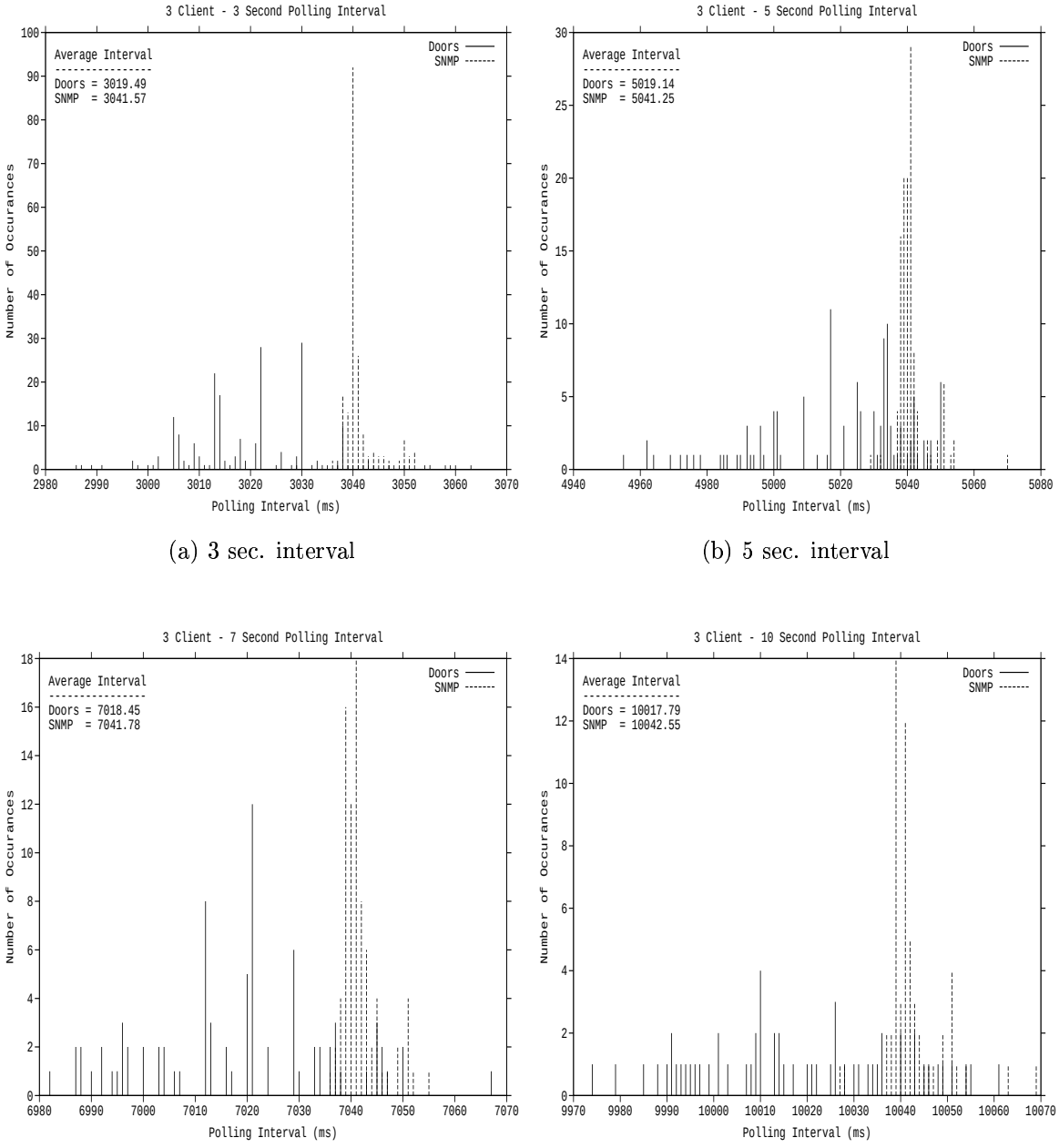


Figure 2.14: 3 client Internet case, standard deviation plot

In the Internet graph Figure 2.14, we see what is expected. The effect on the client has not significantly changed from our isolated network case. The SNMP



(a) 3 sec. interval

(b) 5 sec. interval

(c) 7 sec. interval

(d) 10 sec. interval

Figure 2.15: 3 client Internet case, 3, 5, 7, and 10 second interval histograms

client has the lowest average-based standard deviation, while having the largest actual-based standard deviation. One interesting observation, however, was that the average intervals of the Internet case were actually less than those of the isolated network case. This was because of the 5600 baud clock-rate the routers used in the small isolated network case. To illustrate this difference, a speed comparison for the 3 client case is shown in the following table where the first two columns are the average interval times, and the third shows the ping round trip time.

	Average interval time		Ping
	DOORS	SNMP	time
isolated network	3050.44ms	3130.02ms	54.932ms
Internet case	3019.49ms	3041.57ms	23.218ms

This table shows that it is actually quicker to get information from our testbed machine to the router on the home network than from one side of our isolated topology to another.

2.3.3 Isolated Autonomous System Network Tests

To determine performance comparisons on larger topologies, we implemented our design on an isolated lab configured to represent a small autonomous system (AS) topology, shown in Figure 2.16. This topology involves three logical networks joined by a backbone consisting of three core routers. All of the routers in the AS are connected by Ethernet or serial links and run the dynamic routing protocol OSPF [82]. Hosts are connected to some of the routers through 10MB Ethernet connections.

We compare the amount of bandwidth used when single and multiple clients are present. To measure the traffic incurred across these networks, we monitor the traffic seen by the serial1 interface of core router BB3 (the link on the left side of the top backbone router in Figure 2.16). When standard SNMP methods are used, clients are on n1 and n2 and we poll the Ethernet interface of router P1R2 (target) in Net 1. When the DOORS system is used, clients are on n1 and n2, the naming server and repository both on n3, the SuperServer on n4, and the polling station (n7) close to the target (P1R2) on Net 1.

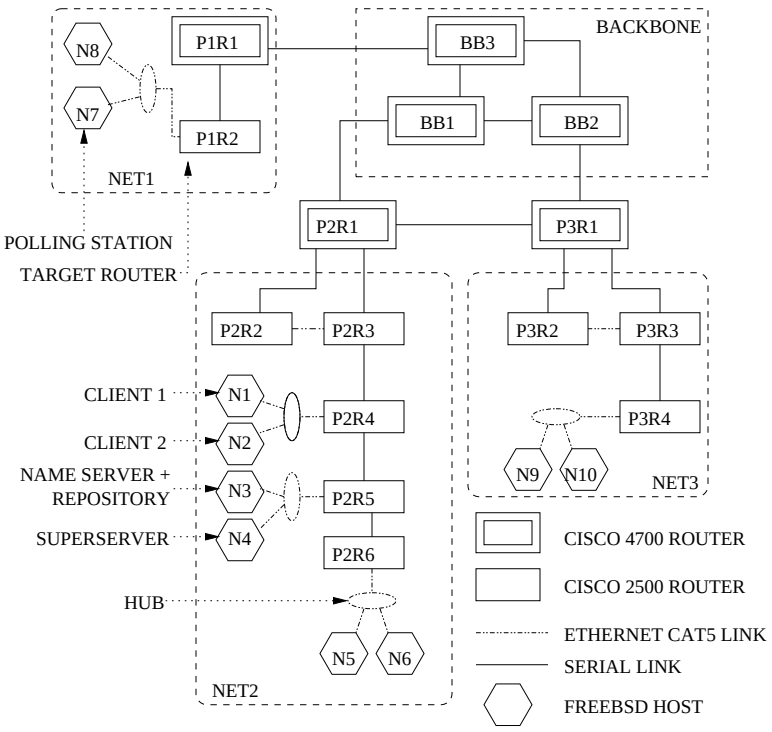


Figure 2.16: Sample Autonomous System (AS) Topology

2.3.3.1 Traditional Data Collection Case

The graph in Figure 2.17(a) shows the bandwidth usage incurred through data collection at three second intervals under both DOORS and traditional SNMP polling in the traditional data collection case. In this graph, the bottom line represents the background traffic of OSPF updates. Because the other plots show similar results, we only present the results from the 3 client, 3 second interval case here. In Figure 2.17(a) we see results similar to those witnessed from the smaller topology; the DOORS system achieves better performance in all cases. Once again, the bandwidth used by the DOORS system remains effectively constant regardless of the number of clients, because only one message per data collection interval is sent back to the repository to be distributed to all clients.

To explore how the DOORS system impacts the client, we use the same standard deviation statistics, average-based standard deviation and actual-based standard deviation (See Section 2.3 for an explanation of these statistics). The graph describing these statistics under different levels of congestion (currently defined by

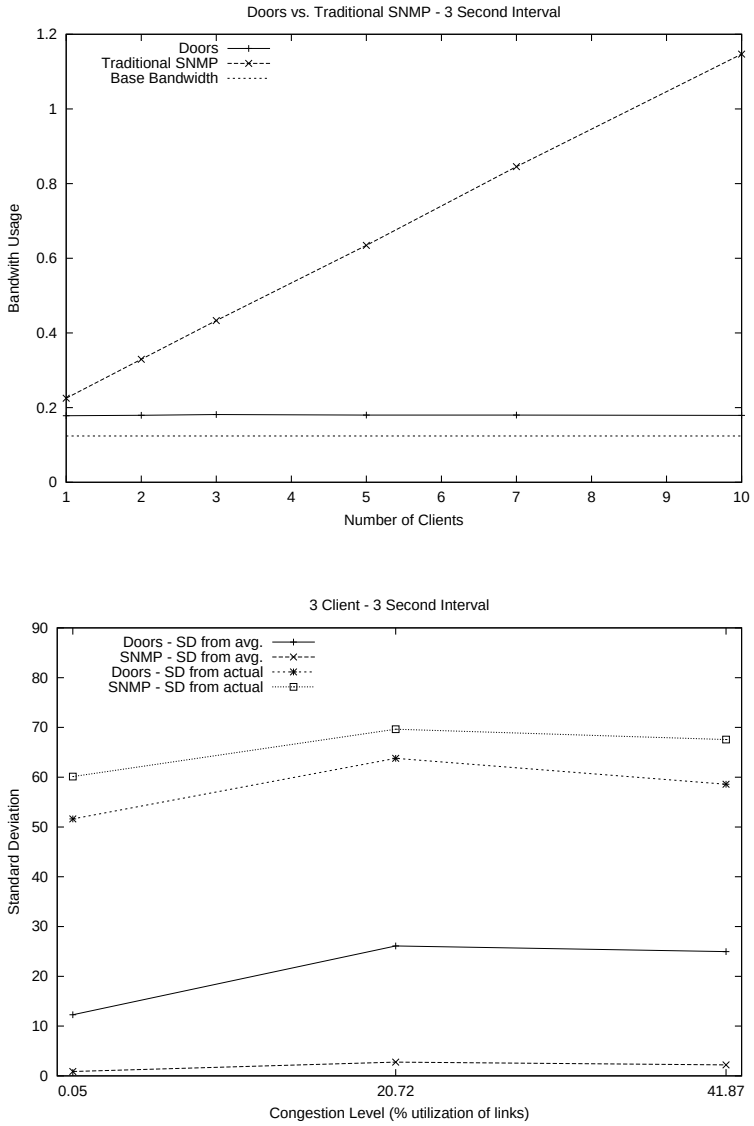


Figure 2.17: 3 client isolated network case, standard deviation plot

average link utilization) is shown in Figure 2.17(b).

Figure 2.17(b) shows the same general trend observed in the smaller topology. Even with the introduction of traffic, the normal SNMP client has a low average-based variance, due to its UDP transport layer. The DOORS system, once again, has a fraction of the latency time that the normal SNMP client has, due to its distribution, and thus has a much lower actual-based standard deviation. As stated before, this means that DOORS may be a bit more variable in the exact times it

returns the data, but it almost always returns it faster than the SNMP client. The other side of this comment is also true. The SNMP client almost always takes longer to return the data, but it returns it at a more consistent interval.

2.3.3.2 Preprocessing Case

We again implemented Thottan and Ji's Network Problem Forecasting solution on our larger AS topology (see Section 2.3.1.3 for an explanation of our implementation of Thottan and Ji's solution). As Figure 2.18 shows, the autoregressive agent uses very little bandwidth, far less than the bandwidth used by standard SNMP polling. In Figure 2.18, the bandwidth statistic of background traffic caused by the routers using OSPF is 0.126. DOORS cannot fall below that point, but it gets close.

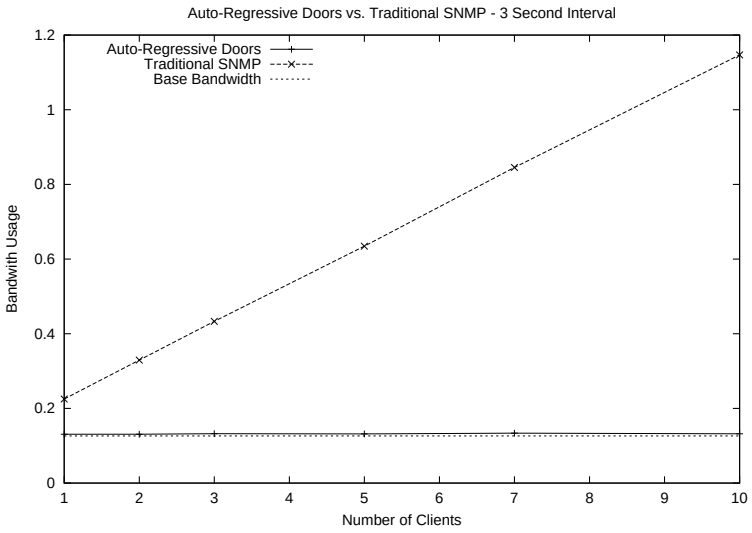


Figure 2.18: 3 second interval autoregressive case, bandwidth usage plot

We do not show standard deviation plots in this preprocessing case because of the difference of magnitude between this and the SNMP traditional polling cases. This difference appears exaggerated when graphed (similar plots are shown in Section 2.3.1.3).

2.3.3.3 Comparisons of all methods

The differences in bandwidth usage across the many methods including our preprocessing case are shown in Figure 2.19. It demonstrates that SNMP clients use

considerably more bandwidth than the DOORS client. It also shows that additional savings are achieved when part of the algorithm is assigned to the agent.

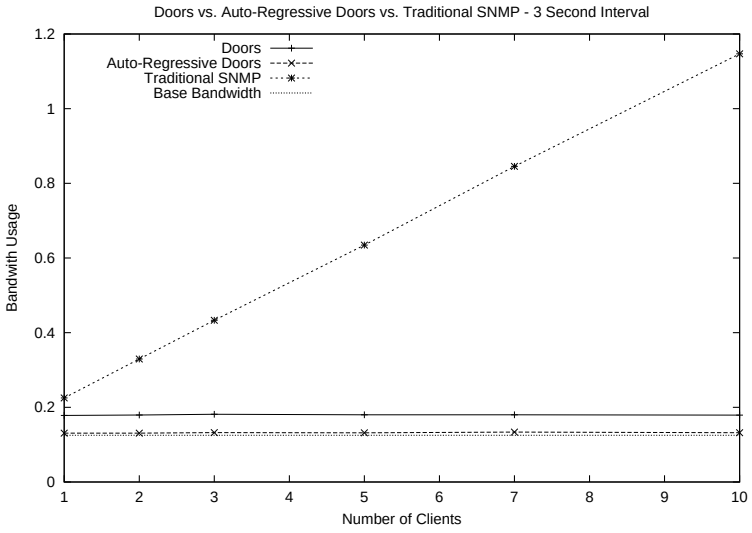


Figure 2.19: Joint bandwidth usage plots of 3 second intervals

2.4 Extended Scalability Analysis Through Simulation

To allow more flexibility in our evaluation and extend the evaluation to larger topologies (that we cannot test in our isolated lab), we simulated each application's behavior using the SSFNet simulator [36]. However, as applications grow in complexity and numbers of connections, the exact behaviors of many simple applications may become very difficult to model. To avoid the hardships of significant simulator code revision, we analyze our application to find appropriate segmentation points whereby we can separate the flow of the application into smaller, easily simulated flows. To support this separation and analyze its validity, we provide an extensive description of *"separable flows"* in Appendix C.

2.4.1 Simulated Autonomous System Topology

In this experiment, we evaluated management frameworks on a simulated topology almost identical to that of the isolated autonomous system used in Section 2.3.3. The traditional SNMP architecture only involves a simple connection

at each iteration; therefore, segmentation of this architecture is not necessary. According to techniques outlined in Appendix C, we segment the frequently occurring parts of the DOORS system into three segments shown in Figure 2.20.

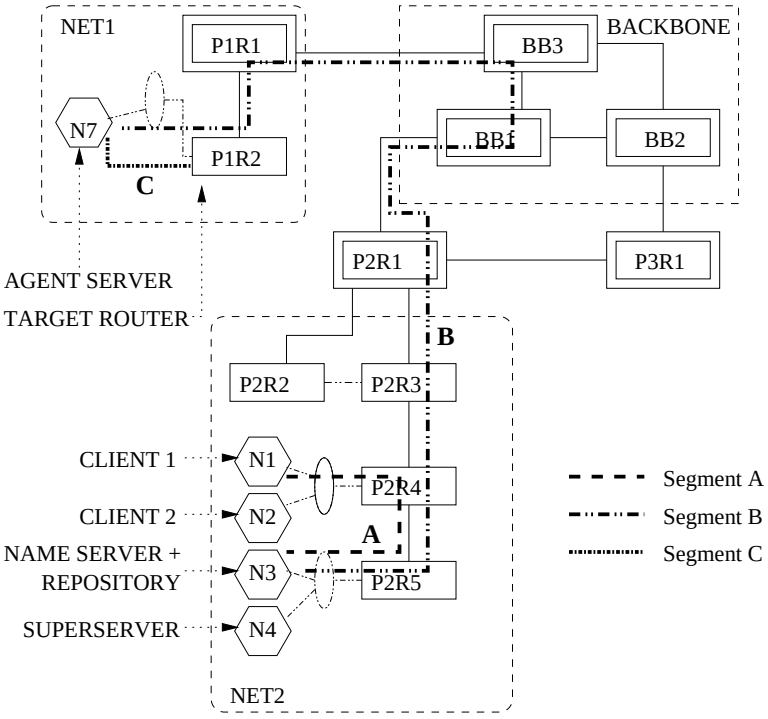


Figure 2.20: Illustration of Application Segmentation

Segment A is the communication between the client and repository involving one request and n responses where n is the number of requested iterations. Segment B is the communication between the repository and the polling station involving sending the agent and n data responses. Segment C is the communication between the polling station and the managed node (router) involving the SNMP request and reply which both happen n times. A simple illustration of this segmentation can be found in Figure 2.21.

In this case, we can describe the interaction, $\bowtie$ (See Appendix C for more information), between these segments as the following:

- $A \bowtie B$ is very small because flow A shares only a small part of the network path with flow B. The network interaction is negligible because the client receives the data after flow B completes its path. The only way the two flows would

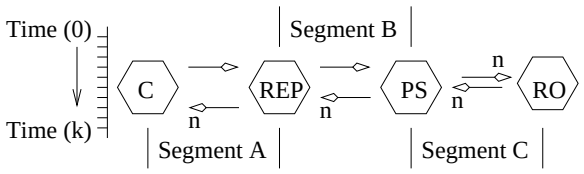


Figure 2.21: Illustration of DOORS connection Architecture

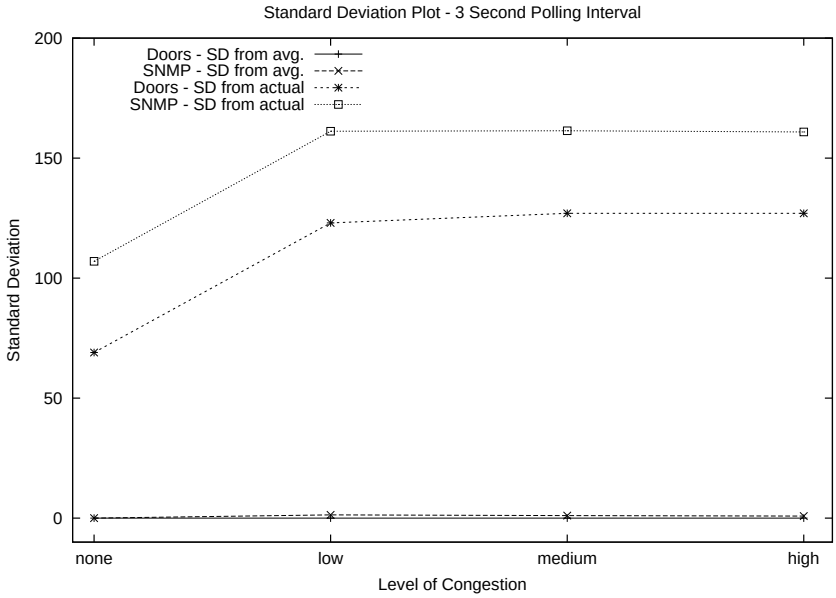
interact is if the one way trip time of flow B's path plus polling interval is less than the one way trip time of flow A's path. The polling intervals are on the order of seconds, while the one-way trip times are on the order of milliseconds, making the above scenario highly unlikely. In the language of Appendix C parameters, if u is not minimal, $\frac{\lambda_{0,Max}}{\lambda}$ will be very low.

- $B \bowtie C$ is also very small. Flow B and Flow C share a small network path. Their network interaction is negligible because they are on the same network and both sections' traffic collectively only consist of one more message per time interval than the traditional SNMP method. Once again, if u is not minimal, $\frac{\lambda_{0,Max}}{\lambda}$ will be very low.

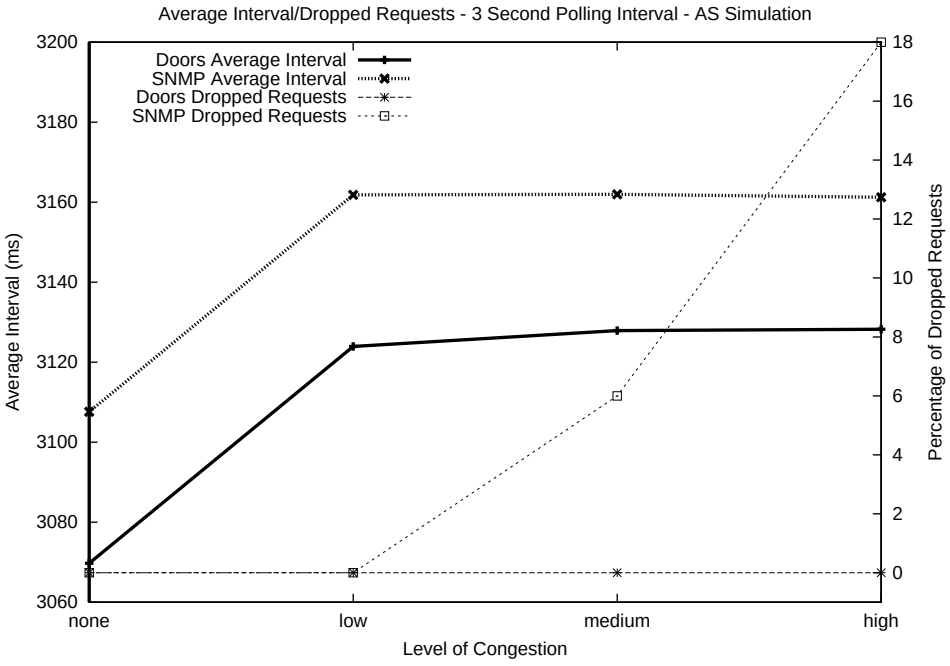
- $A \bowtie C = 0$ because they are in two totally different networks.

Our results are shown in the graphs of Figure 2.22. Figure 2.22(a) describes the deviation in client delay while Figure 2.22(b) explains the average delay and data integrity perceived by the client. In these graphs, we show statistical results of simulations under varying degrees of traffic created by background communication agents sending continuous streams. The congestion levels [none, low, medium, and high] correspond to an increasing number of background communication agents.

In Figure 2.22(a), we see that the normal SNMP client, once again, has the lowest average-based variance, but the difference is not as pronounced as that from Sections 2.3.1.1 and 2.3.3. The DOORS system continues to reduce latency by distributing the monitoring functionality, and thus has a lower actual-based standard deviation. Figure 2.22(b) shows the average inter-polling interval on the left-side y axis, while displaying the percentage of dropped request on the right. A dropped request happens when the client does not receive data during a particular interval



(a) Standard deviation plot



(b) Average interval/packet drops plot

Figure 2.22: AS simulation graphs

due to network load. It is important to note that the traditional method dropped 6% of the SNMP requests in the medium congestion case and 18% in the high congestion case. In the DOORS case, this does not happen because the TCP layer ensures safe arrival of packets.

2.4.2 Simulated United States Topology

To continue our scalability evaluation in an even larger topology, we use a US Internet topology, shown in Figure 2.23, consisting of 25 virtually identical Autonomous Systems. Each AS contains 1,300 hosts, 4 web servers, 27 internal OSPF routers and one AS boundary router running BGP4. The BGP routers are connected with wide area point-to-point links. The wide area topology is a simplified version of the network of one of the largest IP network providers.

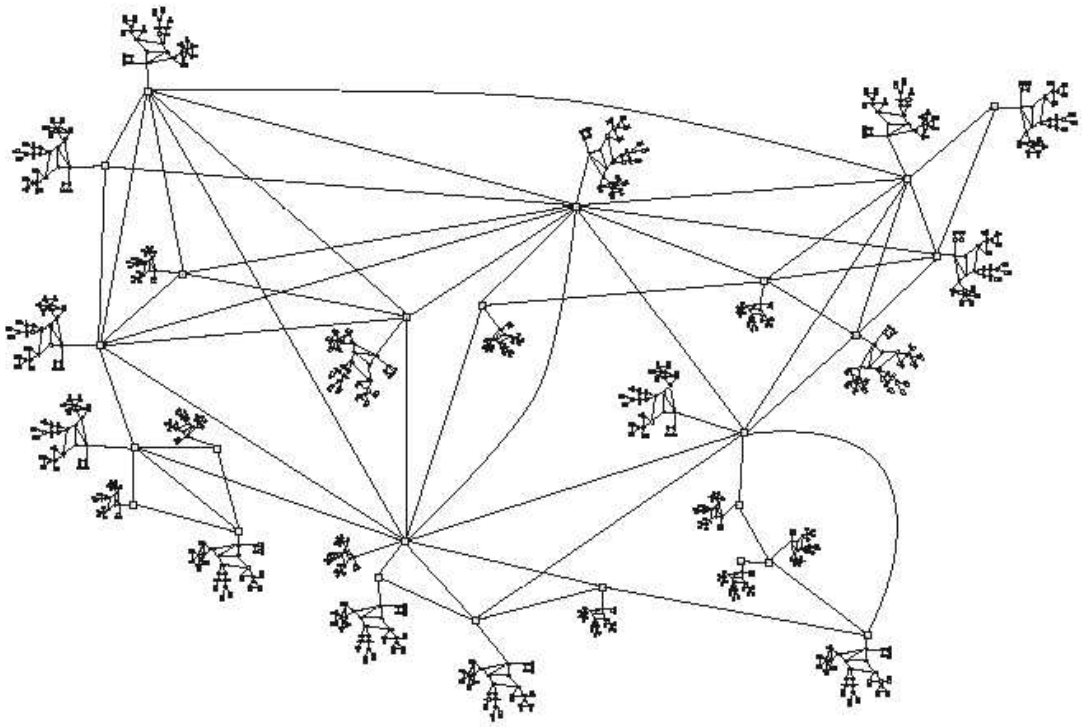


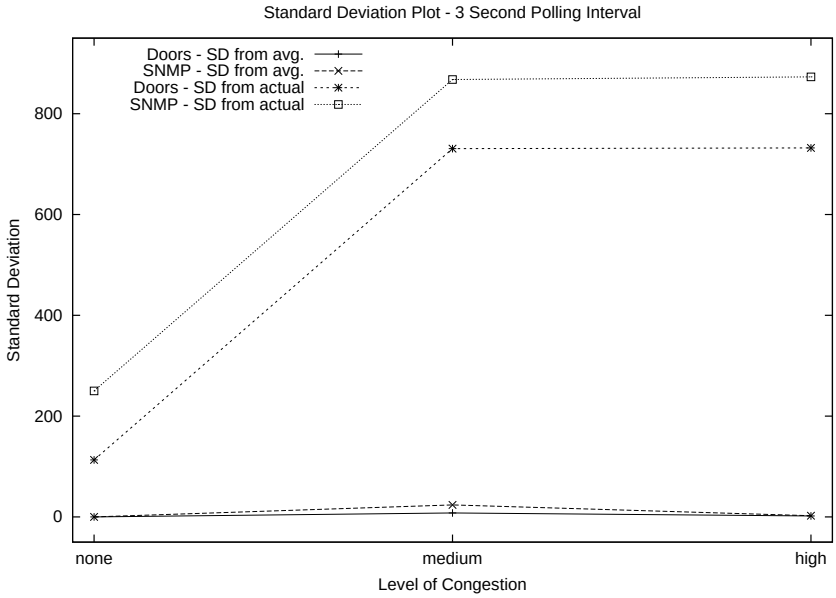
Figure 2.23: Sample USA Internet topology

We simulate our architecture assuming that multiple managers may exist in the same AS with interest in a trouble spot of another AS. A client analysis of

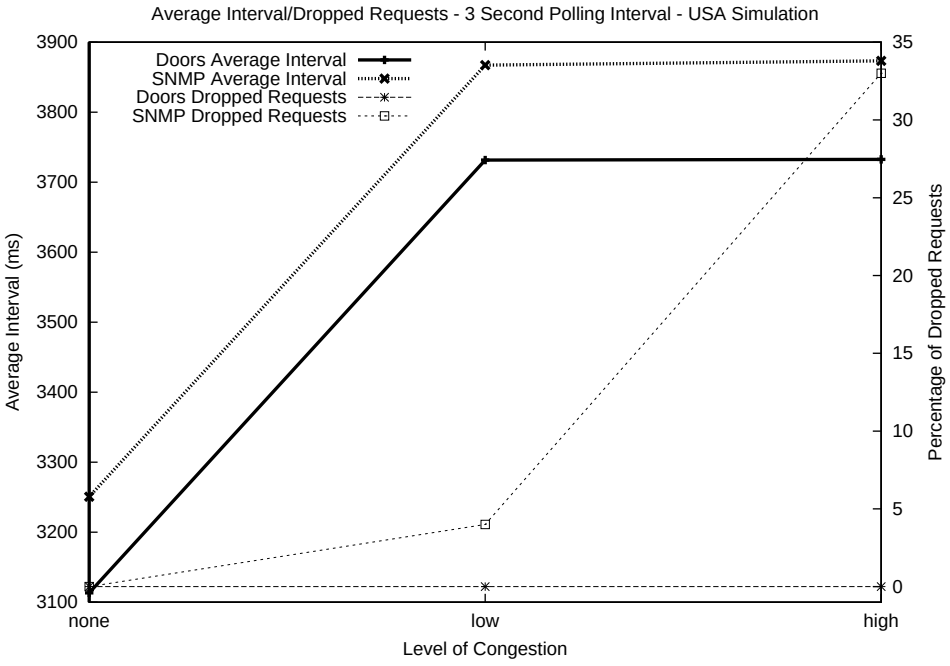
the results is described in Figure 2.24, showing results similar to those seen in the smaller AS simulation (See Figure 2.22). In the USA topology, the traditional method dropped 4% of the SNMP request in the medium congestion case and 33% in the high congestion case.

2.5 Summary

- In a single-client scenario, we see a cost benefit of running DOORS for monitoring network data as compared to conventional SNMP polling methods.
- In a multi-client scenario, DOORS outperforms standard polling methods and this difference grows linearly as a function of the number of clients polling for data. DOORS achieves this advantage thanks to the consolidation of multiple client requests into a single aggregated request.
- DOORS uses TCP connections which make the data transfer inherently reliable as compared to standard SNMP polling methods which use UDP. The added functionality of TCP, comes at the cost of extra bandwidth in the form of added transport layer headers. However, DOORS counteracts this cost by reducing the total number of data messages which pass around in conventional network polling.
- DOORS has proven useful in cases where normal SNMP polling is not feasible, and the management application has no control over the networks in-route to the managed networks.
- DOORS can be extremely effective when encoded with functionality beyond just the simple collection and return of data. When some or all of the algorithm from the client is placed into the agent, we can see large savings on bandwidth and speed of calculation.
- Using DOORS, the client will get its data faster, but may have small deviations in the difference between polls, whereas normal SNMP clients will get the data later than the DOORS clients, but at a more consistent inter-polling interval.



(a) Standard deviation plot



(b) Average Interval / Packet Drops

Figure 2.24: Graphs for USA simulation

We have shown in several distinct scenarios that agent based network monitoring can achieve great benefits at little costs. Distributing network management applications is a necessity as managed networks continue to grow in size and complexity. New uses and applications have caused companies and users to do more with and expect more from their networks. As these uses grow and quality of service guarantees become more wide-spread and popular, the effectiveness of the network management becomes paramount. The corresponding network monitoring without effecting the network traffic is essential for an effective network monitoring application. We believe that a mobile agent approach can solve this problem while providing a framework for administrators to make the management application as proactive as desired by simply equipping the mobile agent with more functionality.

2.6 Related Works

Actively managing networks remains an open topic of research with many challenges. One approach is to have each node receive all the relevant information about the primary traffic as the packets go through the network. Most research and development in this area deals with *Active Networks*. Active Networks allow network packets to contain some code to be computed at each node as the packet travels in the network. According to the concepts of active networks, each packet in the network may contain both data and code. Once one of these "active" packets reaches a node along its path, the node executes the code located in the packet. The code contained in the packet will be executed in an execution environment determined by the node's operating system. This code could consist of function calls, or source code to be compiled in the executing environment. Using active networks, the network can perform customized computations on the data flowing through the network [165].

Some researchers in this field believe that the real power of Active Networks is not in added computational ability, but increased communication ability which focuses on quality of service issues and network traffic classification. Active networks with this ability can monitor and regulate themselves with every packet flowing through the network. They could also ensure that certain traffic would always receive

reliable service [23]. Many times these active networks can be used to augment the protocols used to send the data, providing additional efficiency in the process [3]. Using active networks could certainly enable us to optimize the efficiency of the network. However, the cost of an implementation of such active networks is very high. They require resources at every node in the network to process every packet that flows through the network. An additional delay is also experienced while the component is setting up and breaking down the execution environment. There are also security problems in this approach, because every node in the network will execute the code in the packets, if needed. Some implementations of active networks involve limiting the "active nodes" to those surrounding special links. Many times these are links with historically high congestion probability or the tendency to drop packets. The purpose, in these cases, is to change the traffic (split packets, or send part of the flow through different routes). These implementations are called Transformer Tunnels [10]. Whereas these tunnels do limit the liabilities to a smaller number of nodes, they do not give us the power to look at performance data from other locations in the network.

Work has also been done in the area of Service Level Agreements (SLAs) on IP networks that require information gathering techniques for network managers. SLAs define and enforce agreements for system resources on a service by service basis. Using SLAs and Application Response Management (ARM), information can be gathered about resources used by each ARM-enabled application. Once the necessary information is gathered, resources can be allocated according to the agreements. This can be a very important improvement over traditional methods of network management, as administrators can consider some applications more important than others. SLAs are also used in conjunction with SNMP and other protocols to obtain measurements from a variety of sources. An extensive description of Service Level Agreements can be found in [171].

Barotto et al. [15] designed a network information retrieval tool using Java, CORBA, and SNMP in which web clients use CORBA to submit SNMP requests to an object representing the network. These requests are translated to standard SNMP requests and then retrieved from the managed node. After successful re-

trieval, the values are sent back the the web client. While this project may seem similar, the authors make no efforts to reduce system resources or facilitate network management applications. The system was designed to allow administrators to monitor or configure SNMP parameters from a web browser.

Fuggetta et al. [63], in a case study to show the benefits and disadvantages of different mobile code paradigms, discuss the advantages of an agent-based network management system over the client-server model of SNMP. The authors conduct a mathematical analysis of the network traffic used between SNMP, code on demand, remote execution, and mobile agent systems and determined that, while the effectiveness of code mobility depends heavily on the characteristics of the task, mobile code paradigms such as mobile agents can avoid bandwidth consumption problems in cases when management functionality is most important. Typically, these cases include problem situations where the manager will increase its interaction with the devices and possibly upload configuration changes, increasing the congestion present. Consequently, congestion as an abnormal status, is likely to trigger notifications to the management system, which worsen network load.

Bauer et al. [17] use a repository for management of distributed applications in the MANDAS (Management of Distributed Applications and Systems) project. The authors concentrate on an area other than network management, but there are many similarities with their work and ours. They use a Management Information Repository (MIR) to hold information about their distributed applications. However, they have only one centralized repository. Our implementation will use distributed repositories with advanced communication methods for transferring data between the repositories.

Harista et al. [73] describe MANDATE (MAnaging Networks Using DAtabase TEchnology), which uses MIB (Management Information Base) to support network management. The authors propose to have operators interact solely with their MIB for network management. Their MIB holds information about the network, similar to our network of repositories. Implementation of MANDATE is client-server based with sophisticated client caching. Our implementation is based on distributed repositories and mobile agents to functionally distribute the application.

Authors of [95] focus on real-time management and control of DCE (Distributed Computing Environment) applications using CMIP and the OSI management Framework. There approach is to use RPC (remote procedure call) to communicate with distributed applications retrieving application data and in some cases instituting control measures.

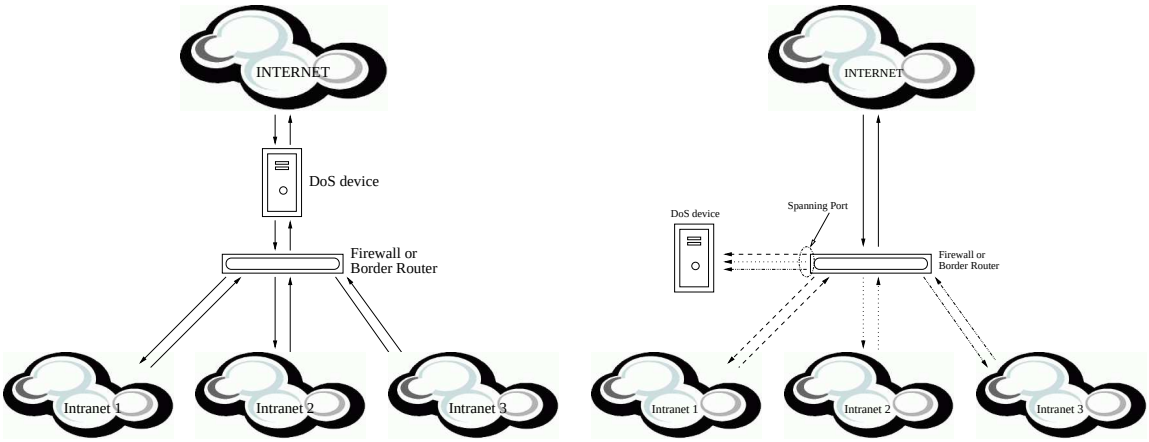
Rajesh Subramanyan and other researchers at Purdue University created the SIMONE system to address the scalability problem through distribution of monitoring tasks [154, 153]. They too use SNMP for management communication. However, they have a two layer approach (through the use of an Intermediate Level Manager), much like RMON but more versatile (the ability to gather more variables, etc.). Our approach can be viewed as taking this technique a couple of steps further, employing mobile agents to facilitate the middle layer and also carry new functionality (including preprocessing and control).

CHAPTER 3
Network-Based Intrusion Detection

Increasing attempts to compromise computer systems by methods such as coordinating distributed attack probes across a network have increased the importance of information assurance and electronic security. Additionally, the foreseen nature of both foreign and domestic future terrorist threats has called for accelerated research and development in securing both commercial and government network vulnerabilities [1]. Among the various solutions that address the prior concerns are intrusion detection systems, software utilities that detect inappropriate or anomalous activity on a computer system or network.

Today's corporations and educational institutions are particularly vulnerable to network attacks. These attacks cause disruptions of service, damage to existing systems, and unauthorized access to sensitive information. As explained in Section 1.1.2, to detect attacks from the network, intrusion detection systems (IDSs) use some form of network sniffed data. In enterprise networks, the area of sniffing interest is usually at the entry points of the organization's network. Once placed in promiscuous mode, a machine's network interface can read all network traffic that goes across the same shared media. This raises concerns in modern switched or highly segmented environments where the shared network segment is very small or non existent [117]. In the case of switched environments, many of the modern-day switches ship with spanning ports (also known as mirroring, or tapping ports). These ports simply repeat all traffic from the normal switched ports onto the spanning port. Examples of where an IDS would be placed in an enterprise network are shown in Figure 3.1(a) and Figure 3.1(b).

In this chapter we will present some of the current industry standards used to address the network-based intrusion detection problem followed by two of our own methods. The first of our methods is a misuse detection system using finite automata while the second uses neural networks to merge aspects from misuse and anomaly approaches (recall the difference between anomaly systems and misuse



(a) Inline Network-Based Intrusion Detection

(b) Wiretap Network-Based Intrusion Detection

Figure 3.1: Network-Based Intrusion Detection Architectures

systems described in Section 1.1.2). Lastly, we will review other research efforts in this area.

3.1 Current Standards

The first steps to stop network-based attacks were systems which stood in front of the protected network only permitting expected requests to go pass through. These machines are called firewalls. However, corporate sites have had to open so many holes in their corporate firewalls to support mobile workers, telecommuters, business partners, and suppliers that the once easily defined network perimeter is now difficult to recognize. To provide high level of access, companies must hide data from the firewall by using technologies like encryption and VPN services [175]. Therefore, more intelligent systems are needed to accompany today's firewall technology.

There are many network-based intrusion detection systems. Most monitor network traffic by passively listening to the network as either a node using the shared medium or a node in the direct path to the network. These products typically use software packages like Sun's "netlog" or the more popular "tcpdump" package from

the Lawrence Berkeley Laboratory which places a network interface in promiscuous mode and listens to all of the passing network traffic. Other hardware network-level analyzers are also available from manufacturers of these tools. They typically have a set of rules that incoming traffic is compared against for attack detection. Some corporate organizations with products using these types of strategies are NFR Securities [124], Symantec [157, 135], and ISS [87, 86]. Some of these devices also attempt to model normal activity, causing varying levels of alarm when deviation from the normal base is detected. Extensive reviews and comparisons of current commercial network-based intrusion detection methods can be found in [7, 122].

3.2 Time Dependent Finite Automata for Denial of Service Detection

3.2.1 Introduction

Our first network-based IDS falls under the category of misuse or penetration identification. After a "signature" is defined that identifies a manifestation of an attack, the attack can be discovered in the monitored network traffic. In case of our system, the signature of an attack is defined in terms of sequences of system events and traffic data, with which we can attempt to identify these attacks in real-time to prevent any further damage to the system. Furthermore, by knowing precisely which attack was detected, we can possibly tailor defensive strategies and/or raise appropriate system alarms.

In the approach described in this section, we focus on detecting one type of computer attack: denial of service (DoS). Generally, DoS attacks are classified as a characteristic set of events (e.g., an intentional flood of unwanted traffic) that diminishes or eliminates a network's capacity to perform its expected function. As a result, the compromised system looses the ability to handle legitimate traffic/requests, experiences a loss in available link bandwidth, and is overall unable to provide normal service to its users. In many cases, service is denied until the attacker's address is discovered so that further traffic from that source can be blocked. However, often the source address is spoofed so that the true origin of an attack becomes untraceable. A more difficult scenario arises in cases of the distributed denial of service

(DDOS) attacks, in which multiple clients are coordinated to simultaneously flood a victim machine or network with requests. In this case, more resources on the targeted system could be in jeopardy and more clients need to be blocked to recover from an attack. As with other attacks, it is most effective to detect DoS (and DDOS) attacks as early as possible.

This work contributes to the detection of DoS attacks primarily by exploiting the time-dependent characteristics of DoS attack behavior. Our method and framework compose a computationally efficient, non-obtrusive, and scalable solution addressing an important subset of network security. In this section, we will discuss our specialized automata-based approach to misuse identification, as well as our motivation for its use. Following that discussion will be an overview of the system's architecture, and an evaluation of the system's test results using the publicly available datasets from both the 1998 and 1999 Defense Advanced Research Projects Agency (DARPA) intrusion detection evaluations [109, 110].

3.2.2 The Time-Dependent Deterministic Finite Automata Approach

To explain time-dependent deterministic finite automata structure we must first explain the more simple deterministic finite automata. We will then explain the time-dependent addition followed by an explanation of its applicability to DoS attack representation.

3.2.2.1 Deterministic Finite Automata

Conceptually, a deterministic finite automaton (DFA) is an abstract computational model designed to represent an idealized computer. Just as a computer changes states of operation and produces some outputs given particular inputs, so does a DFA. Specifically, DFAs are designed to recognize or accept member strings of a particular regular language [144]. In our case, we want to recognize the language of DoS attacks (to be explained in Section 3.2.3.2). DFAs possess two important properties: they embody a finite number of states and, they are deterministic, meaning that given a current state and an input, the automaton "transitions" to only one state (which could be the same state or a new one).

Formally, a DFA is defined by:

1. A finite set of input characters (its alphabet)

2. A finite set of states, with one state defined as an initial state and a subset of states distinguished as final states

3. A set of transitions that is a Cartesian product of the "set of states" times the "alphabet" times a "subset of states"

Hence, in referring to the third component of the definition, each transition is a triple: originating state, transition character, and target state. This represents a move to the target state that the DFA undertakes when it receives the transition character while in the originating state. Informally, DFAs are represented by state-transition diagrams. Circles represent the automaton's different states and unidirectional arrows labeled with input characters represent transitions between different states triggered by the specific character in the input. Typically, final states, in which the entirety of an input string is accepted, are double-circled. Also, according to the formal definition, for every state and every input character, there must exist an exiting transition. Figure 3.2 is an illustration of a simple DFA which accepts strings from the alphabet {a, b} of sequence "aba."

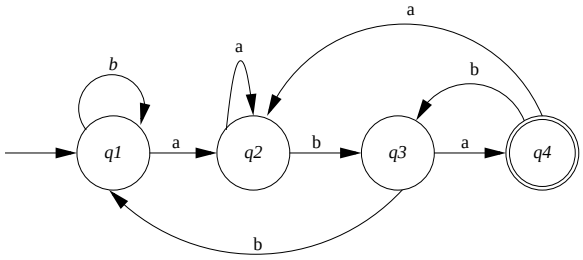


Figure 3.2: An example DFA

In this example, q1 is the start state and q4 is the final state. If the DFA is in state q1 and receives "a" as a current input character, then the DFA moves to state q2. If in state q2 the DFA receives "b" as an input, it moves to state q3. If an "a" is received by the DFA in q3, the automaton moves to the final state and the input string is accepted. An examination of Figure 3.2 shows that any input disrupting the pattern "aba" moves the DFA back to a state where it can continue to monitor

for the desired substring. While the DFA in Figure 3.2 has only one final state, this is not a requirement for DFAs in general.

3.2.2.2 Time-Dependent Deterministic Finite Automata

The time-dependent deterministic finite automaton (TDFA) is an extension of the machine we just described. This extension enables it to consider more than just the sequence of input characters; TDFAs also consider the time intervals between receiving input characters in recognizing members of a language. This becomes very beneficial in the use of automata to recognize DoS attack signatures since many DoS attacks are dependent upon the time intervals between arriving network packets. Figure 3.3 shows an example TDFA. We can think of this machine as recognizing the pattern "$a, b < 5, a < 5$." In other words, the "b" must occur within five seconds of the initial "a", and the last "a" must occur within five seconds of the "b". All transitions shown without a time restraint are default transitions. If the desired input arrives, but not within any specified time constraint, then the default transition for the desired input is used. Notice that in our example (Figure 3.3), the transition from q1 to q2 is a default transition even though it takes the TDFA closer to the final state. This is because the currently received "a" may be the first character of the attack pattern; it does not make sense for it to come within five seconds of some other input.

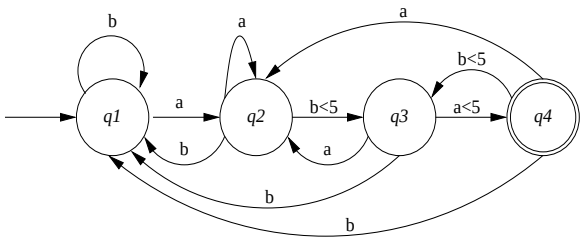


Figure 3.3: An example TDFA

A TDFA, as defined here, is a non-trivial extension of DFAs, because the input to the TDFA is a pair consisting of a character from the input alphabet and a time of this character arrival (which we currently represent as a real number). The Cartesian product of these two elements of the TDFA input yields an infinite

alphabet, so a TDFA cannot be represented as a product of two DFAs. However, a TDFA uses the actual difference of arrival times between two input characters (this still leads to an infinite set of possible differences). Furthermore, TDFAs use these differences to create boolean values resulting from the comparison of each difference to constant or computed variables defined in the TDFA, which finally yield the finite input to our automata. Hence, formally, a TDFA can be represented as a combination of a token arrival time preprocessor and an ordinary DFA. The token arrival time preprocessor simply yields a boolean value to each inequality defined in transitions exiting from the current state of the TDFA. The states of this particular DFA are the same as those of the TDFA. Each transition in the TDFA with a time constraint is represented by the DFA transition labeled with the same character and a true boolean value. Each transition in the TDFA without a time constraint is represented by two transitions in the DFA, one with true and the other with false boolean values.

3.2.2.3 DoS Attack Representation Using Time-Dependent Deterministic Finite Automata

The very nature of TDFA models makes them a logical choice in representing DoS attacks. DoS attacks frequently manifest themselves as a characteristic series of network events or special packets that progressively render a particular resource inoperable. Therefore, we use the transitional arcs of TDFAs to represent those characteristic attack events. TDFA states are used to represent incremental conditions of a system as it reaches a state of full penetration. The final states of a TDFA then represent states of attack completion.

We observed that the time scales of the various DoS attacks differ dramatically. After analyzing a week of intrusion data from the DARPA dataset, we derived a histogram of the attack durations over a period of several days (see Figure 3.4). In this figure, because of the large variance of the attack durations, the x-axis is an approximate logarithmic scale of seconds while the y-axis represents the duration count.

While many of the attacks take place over a very small time period, a signifi-

cant number of attacks take over a minute and quite a few take several times longer. DoS detection systems must cope successfully with this wide range of time scales. Incorporating a time component into an automaton structure gives us the ability to use temporal differences for accurate detection.

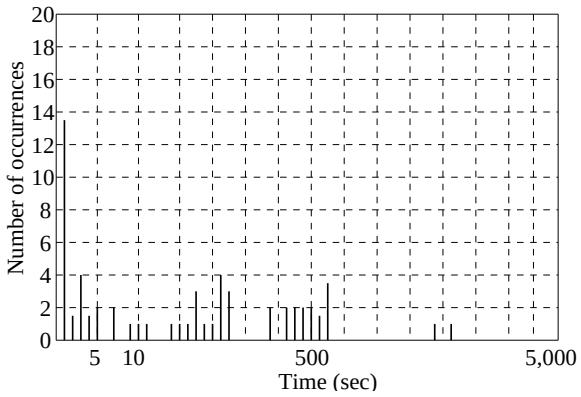


Figure 3.4: Histogram of attack durations from 10 days of attack data

While detection accuracy has traditionally served as a common benchmark for IDSs, the possible circumstances of detection "inaccuracy" should also merit the attention of the intrusion detection community. A recent study showed that on average, various commercial IDSs repeatedly "crashed" under the burden of reporting too many false alarms from an ISP's network traffic. The study illustrated that frequent amounts of either low priority or false alarms can cause an IDS to totally miss significant intrusions for which signatures are defined [122]. This definitely serves as motivation for the use of more intelligent attack signature models that intrinsically respond to only the events and variations defined by the author of the signature.

Another problem common with most IDSs is the lack of the ability to perform detection on high-speed links. This makes most network-based IDSs situated only for detection coverage on an individual node rather than across an entire network. This concern is currently being addressed by the development of more intrusion detection appliances with specialized hardware designed to reliably inspect high-speed IP traffic [119]. Even with the advent of more robust hardware, poorly designed inspection algorithms can still throttle detection performance, leaving software as the bottleneck. Because automata-based models are traditionally efficient in com-

putation, we are confident that our proposed detection method will complement intrusion detection appliances used for high-speed network traffic.

3.2.3 System Architecture

In this section, we discuss the full architecture of our system, including its prime functions and its various subsystems and their interactions. Before continuing, it would be useful to note that our system detects DoS attacks occurring in TCP, UDP and ICMP network traffic. These are all industry-standard network communication protocols [151]. The following are the key characteristics of our IDS:

- Our system detects DoS attacks from both real-time and historical data
- It uses time-dependent deterministic finite automata to model and confirm attacks
- It supports the updating of attack models without interruption to other system components

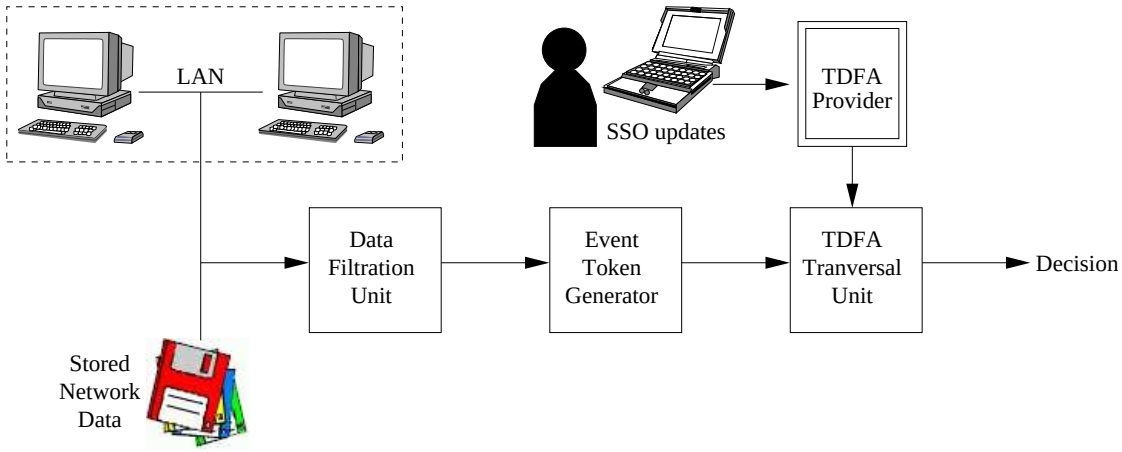


Figure 3.5: Overview of system architecture

Having an option of operating from real-time or historical data offers a couple of advantages for a site security officer (SSO). Real-time monitoring of network traffic provides the best level of protection because ongoing attacks may be stopped. However, with the addition of recorded (historical) datasets, off-line operation still

allows the SSO to see if and when attacks might have occurred while the IDS was down for maintenance. Also, the SSO can use experimental datasets to test and tune the system for newer attacks. Next, while confirmation procedures will be discussed in greater detail later, it is important to further note the significance of time-dependent deterministic finite automata in our system here. Earlier, we discussed how TDFAs serve as an appropriate model for DoS attacks. However, the additional benefit of TDFAs for site security officers is that they permit the storage of both an attack's base signature and its variation(s) using only one model construct. A SSO can even use one TDFA model to represent multiple DoS attacks, as we do in our implementation. Additionally, using time-based information in attack signatures increases the accuracy of detecting DoS attacks. Lastly, the ability to update attack models without disturbing other system components prevents degradation of the system's performance during detection. As will be discussed later, this feature also presents the opportunity for a distributed architecture.

At present, four distinct components make up our system:

1. The data filtration unit
2. The event token generator
3. The TDFA transversal unit
4. The TDFA provider

External components with which our IDS interacts include a local area network (LAN), stored (historical) network traffic data, and a SSO client machine. The connectivity of the entire system is presented in Figure 3.5.

3.2.3.1 Data Filtration Unit

Network packets carry a wealth of information (i.e., sequence number, header length, checksum, etc.), some of which may not be needed for the purposes of network-based IDSs. Since our focus is on DoS attacks, the packet data fields in which we are interested include those pertaining to such information as source and destination addresses and the various flag fields (i.e., SYN and ACK). The function

of the data filtration unit (DFU) is to process relevant network packet information for subsequent components of the system (according to the IDS flow of data). As seen in Figure 3.5, network traffic data originates from either of two sources: a local area network (LAN) or a stored data source. The utility module that we use to gather the data is tcpdump [88], which can provide a record of network activity for a particular machine in ASCII text form, delimited into various fields. The DFU can be manually configured to parse whatever tcpdump fields the user requests. Table 3.1 lists the fields that we chose to extract for our prototype.

Packet type	Timestamp
Source IP address	SYN flag
Destination IP address	ACK flag
Destination port	Echo request
More fragments flag	Echo reply
Fragment offset	Sequence first
IP length	Identification

Table 3.1: Current parsed tcpdump fields

Live data originating from connected LAN devices are used in real-time detection while stored data, our second source of extraction, is used in offline mode. As opposed to the prior source, stored data usually resides in a log file and may be in either ASCII or binary form. If the data filtration unit receives the process packet information in binary form, it first converts it to ASCII text for further processing. In this case, the associated overhead of converting binary data into ASCII form constitutes an irrelevant performance hit since we are most concerned with penetration threats present in live, not stored, network traffic. Regardless, the end product is a delimited ASCII text message which contains specific network event information (including TCP, UDP, and ICMP packet data).

3.2.3.2 Event Token Generator

After network event data are processed by the DFU, the corresponding ASCII text (remaining in a delimited format) serves as input for the event token generator (ETG). The ETG is then responsible for translating the DFU text messages, each

representing a particular network event, into special tokens. We must note that a relationship between DFU messages and ETG tokens is not necessarily "one-to-one." It is quite possible that the information in one message will cause the ETG to generate a sequence of tokens. Also, we have defined only a finite number of ETG tokens, each corresponding to network events we observed as being characteristic of DoS attacks. Therefore, many DFU text messages might be ignored by the ETG. All of the predefined ETG tokens, each being a string of one or more ASCII characters, compose an alphabet used by our system for recognizing DoS attacks. The efficiency of domain-independent IDS languages has been shown with such languages as STATL [51] (used by both USTAT and NETSTAT), which partially serves as our motivation for using a proprietary language. Our other incentive is that it simplifies the definition of our detection engine: the TDFA transversal unit, which is discussed in the next section.

Token(s)	Definition
S	Packet's SYN flag is checked
F	Packet's MF flag is checked
J>5	Non-initial ACK packet; time interval between this and current packet is greater than 5 seconds
&	First ICMP echo reply packet to a particular destination [address and port]
B<60	Not-initial UDP packet to echo port of particular destination; time difference between this and previous is less than or equal to 60 seconds

Table 3.2: Sample ETG tokens and their definitions

The following is a condensed explanation of how the ETG operates. Suppose that after reading the DFU text message, the ETG determines that the UDP destination port number of the packet is set to an echo port number (a condition of a possible UDP Storm attack). Subsequently, the ETG generates the token "e" and sends it to the TDFA transversal unit, where it will be used for UDP Storm attack recognition. Table 3.2 highlights just a few of the tokens that compose the language for our system.

Attack Name	Protocol	Effect
Land	TCP	Operating system loops and eventually freezes
SYN Flood	TCP	Legitimate service requests are denied as CPU resources become totally consumed; operating system may crash or loop
Ping Flood	ICMP	Network slows down; network connectivity may be disabled
Process Table	TCP	Process table is completely filled with network server instantiations; new processes cannot be started
Smurf	ICMP	Host floods both itself and intermediate network with ICMP echo replies
Teardrop	N/A	Host may hang or crash
UDP Storm	UDP	Legitimate service requests are denied as CPU resources become totally consumed; network may become congested

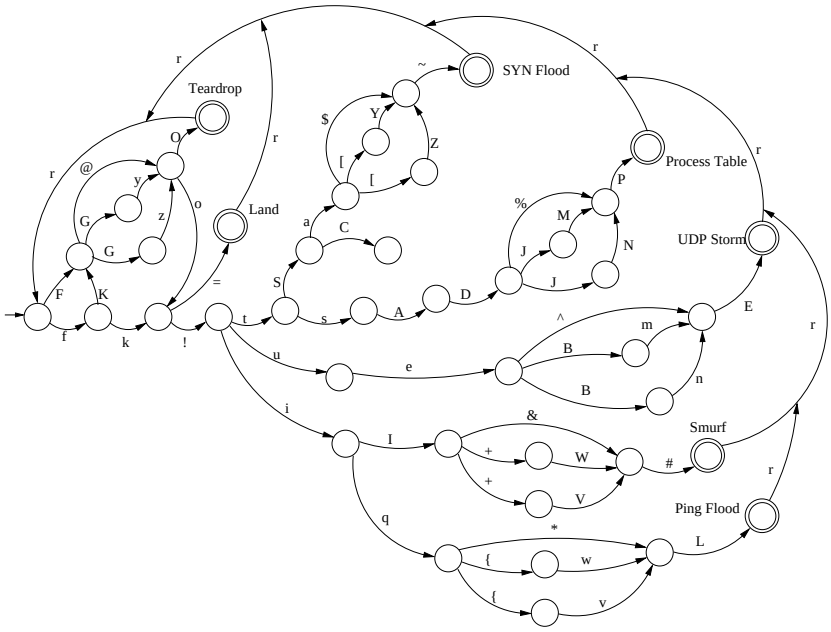
Table 3.3: Current DoS attacks for which we have written TDFA signatures

3.2.3.3 TDFA Transversal Unit

Most intrusion detection systems have some distinguishable core component primarily responsible for recognizing attacks. In our system, the TDFA transversal unit (TTU) acts as the main attack detection engine. The TTU is what actually embodies the user-defined TDFA that represents the various DoS attacks.

The relationship between the TTU and the previous module, the event token generator, is best thought of as that between a physician and patient. The ETG (patient) displays "symptoms" (on behalf of the guarded host) of probable DoS attacks. As discussed earlier, these symptoms materialize themselves as tokens. The intent of the TTU (physician) is to read the tokens and "diagnose" the host as being under, or not under a state of attack. It does this by using the ETG tokens as input characters to traverse the supplied TDFA. In our current prototype, when the TTU finds that its TDFA has reached a final state, it alerts the site security officer that an attack (specified by its respective final state) has occurred.

As mentioned earlier, it would be beneficial for a SSO to verify the effectiveness of the IDS using recorded network traffic data containing traces of successful



(a) Entire TDFA model

(b) Smurf attack portion of the TDFA

Figure 3.6: Experimental TDFA model with a detailed view of the smurf attack showing time constraints

intrusions. Specifically, this entails verifying the correctness of the TDFA transversal unit as well as the TDFA structure. To gain insight as how to design the most effective TDFA models, we inspected attack-laced network traffic data provided by the DARPA intrusion detection evaluation. Since this data would be used for our testing purposes, it was an ideal place to start defining various DoS attack signatures, especially the time-constraints of those signatures. Additionally, we consulted

Kendall's work, which contained signature descriptions of the attacks used in the DARPA 1998 evaluation [96]. We also included variations of the attack signatures that would likely cover multiple incarnations of certain attacks due to varying time durations or distinct packet characteristics. Again, this highlights the added advantage of using a TDFA to model attack signatures. Figure 3.6(a) is an abstraction of the TDFA we used for experimental purposes. Figure 3.6(b) shows a more detailed portion of the TDFA representing a Smurf attack, complete with time constraints on some of the transitional arcs. Notice the lack of default transitions in the example TDFA. We will point out later how system administrators need only to define "linear" attack models, devoid of default arcs. Table 3.3 lists seven DoS attacks that we designed our current TDFA to recognize.

3.2.3.4 TDFA Provider

We mentioned earlier that our system supports the updating of TDFA attack models without interruption to the rest of the system. The TDFA provider makes this feature possible. When a site security officer (SSO) wants to replace the resident TDFA, it interacts with the TDFA provider and gives it a description of the new TDFA model. However, models such as those depicted in Figure 3.2 and Figure 3.3 are not required. A client has only to specify the attack "signature," meaning only those states and transitions that lead directly to attack acceptance (recognition). All other transitions, such as those moving a TDFA back to its start state, would automatically be added by the TDFA provider. This is convenient for the client as he or she needs only to provide a simple, linear attack model. The TDFA provider then supplies the TDFA traversal unit with the user-defined TDFA description. The updating of the previous TDFA with the new attack model can occur without degradation to other components of the system because the TDFA provider interacts only with the TDFA transversal unit. It should be noted that at start-up time, the TDFA transversal unit contains a default TDFA model, so detection is possible before interjection by the SSO. As will be described with more detail in Section 7.2.2, this component makes it possible to develop a distributed modular architecture permitting a more automated approach to updating attack signatures.

Dataset	Attack	DARPA Time	TDFA Time
1998_week4_tues	SYN Flood	11:55:38	8:50:15
	Ping Flood	20:11:31	20:11:31
	Teardrop	23:15:08	23:15:08
1998_week5_mon	Teardrop	08:15:02	8:15:02
	Smurf	12:53:15	12:53:15
	Smurf	15:33:28	15:33:28
1998_week5_fri	SYN Flood	17:27:07	17:27:07
	Smurf	18:00:15	18:00:17
1998_week6_tues	Ping Flood	13:04:56	13:04:56
	Land	17:53:49	17:53:49
1998_week6_fri	Teardrop	08:32:12	8:32:12
	SYN Flood	09:31:52	NO
	Smurf	19:12:37	19:16:27
1999_week2_mon	Ping Flood	08:50:15	8:50:15
	Land	15:57:15	15:57:15
1999_week2_thur	SYN Flood	11:04:16	11:04:16
	Land	15:47:15	15:47:15
1999_week2_fri	Ping Flood	09:18:15	9:18:15
	SYN Flood	11:20:15	11:20:15

Table 3.4: Test results from DoS attacks in 1999 and 1998 DARPA datasets

3.2.4 Test Results

As stated before, we used datasets from the 1998 and 1999 DARPA intrusion detection evaluations in testing our system. In total, we used eight tcpdump training data files: five from 1998 and three from 1999. Currently, we have tested our system on all but two of the seven attacks that we listed earlier (UDP Storm, and Process Table attacks were not present in the data files that we used from the DARPA web site). The results of the test are presented in Table 3.4. The third column shows the time of the specified DoS attack, according to the DARPA dataset. The fourth column shows when our system recognizes the timestamp of the packet finalizing the specified DoS attack. As is seen in Table 3.4, we were very accurate in detecting various DoS attacks. However, in one dataset, 1998_week6_fri, we failed to detect the SYN Flood attack (failures are highlighted in Table 3.4). The attack was successfully detected in most other datasets, although there was also a problem

with the 1998_week4_tues dataset. In this case, we prematurely detected the SYN Flood attack approximately three hours before it supposedly occurred. This case is actually a "false-positive" at 8:50:15 and a "false-negative" at 11:55:38. More detailed information on this attack is needed to see if indeed the first indication of the attack appeared as early as our IDS reported.

To compare our results with other IDSs on similar datasets, we examine the collective results from the DARPA evaluation documents for the 1998 and 1999 attacks. Because our system only functions on DoS attacks for which signature automata have been created, we only focus on the results others have obtained on old (or previously seen) DoS attacks. One should note that different IDSs and different attacks were used in the 1998 and in the 1999 evaluations. The five attacks on which our TDFA were tested constitute a subset of those tested in the evaluations. A small graph summarizing the collective results from the 1998 and 1999 DARPA evaluations is given in Figure 3.7. While not perfect, our current results are very competitive when compared to the results achieved by the systems used in the DARPA evaluations.

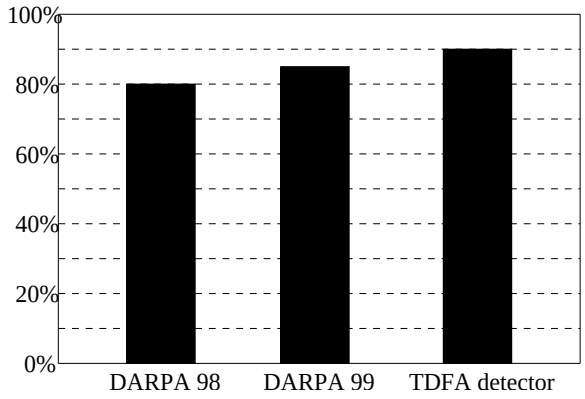


Figure 3.7: Graph summary of detection results from DARPA 1998 and DARPA 1999 evaluations for DoS attacks

3.2.5 Conclusion

Although the current results from our system evaluation are encouraging, we are concerned about the lack of SYN flood attack detection in the 1998 datasets. This could have been caused by our TDFA not embodying all variations of this

attack. We are currently studying the DARPA datasets in attempts to improve our TDFA model. We are also in the process of testing our system for the remaining attacks present in the DARPA datasets: process table and UDP storm.

We also acknowledge that our IDS, being a misuse detection system, has a significant flaw common to all such systems: it can only detect attacks for which it knows a signature. This type of IDS is most useful as a attack filtering tool for other systems which may be able to detect unusual or uncharacteristic behavior.

In general, network-based intrusion detection is still in early stages of development. As networks increasingly become more complex, the need for sophisticated security tools will rapidly grow in importance. While our TDFA IDS does not detect all malicious network penetrations, it does a great job detecting a significant subset of these attacks: denial of service. DoS attacks are widely publicized as they inflict damage on many substantial commercial, educational, and government web sites. Our TDFA IDS detects these attacks in an accurate and efficient manner and is compact enough to be coupled with other IDSs (perhaps even anomaly detection systems) to build a complete suite of general attack detection/prevention tools on multiple platforms.

3.3 Artificial Neural Networks for Denial of Service Detection

In many cases, DoS attacks involve a specific trend of traffic designed to render a particular resource incapable of servicing legitimate users. We believe that denial of service and other network-based attacks leave a faint trace of their presence in the network traffic data. Therefore, we designed an anomaly detection system that detects network-based attacks by carefully analyzing this network traffic data and alerting administrators to abnormal traffic trends. It has been shown that network traffic can be efficiently modeled using artificial neural networks [8, 27]. Therefore, we use Multi-Layered Perceptron (MLP) neural networks, a supervised learning method, to examine network traffic data. In our system, it becomes necessary to group network traffic together to present it to the neural network. For this purpose, we use a special type of artificial network called a self-organizing map (SOM), an

unsupervised learning method. SOMs have been shown to be effective in novelty detection [183], automated clustering [125] and visual organization [99].

3.3.1 System Overview

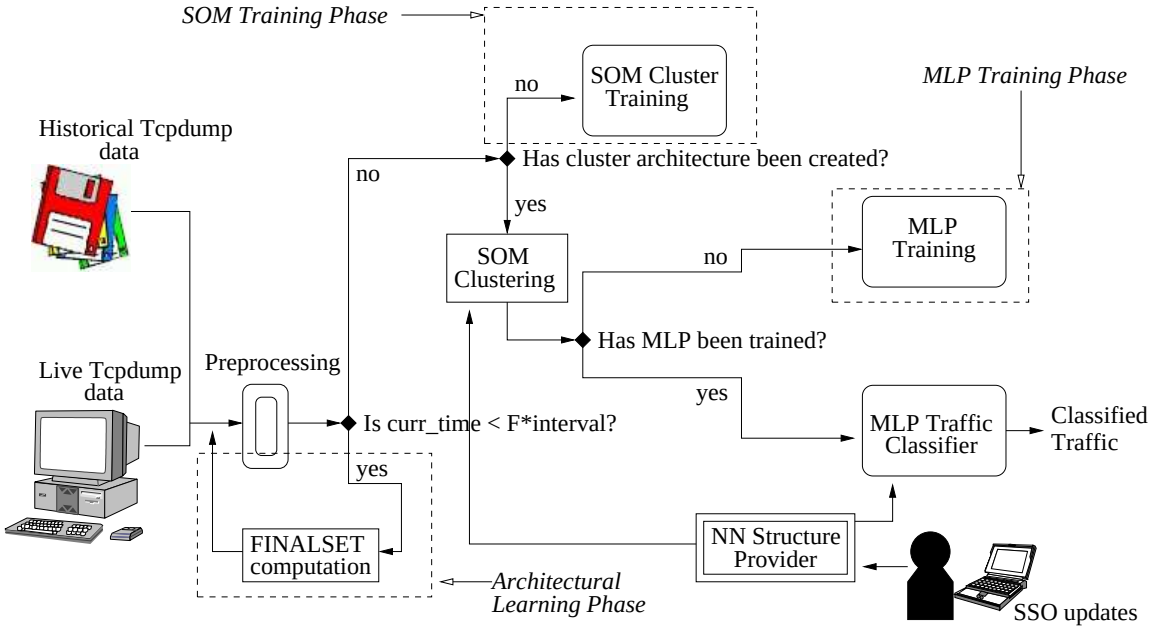


Figure 3.8: Neural Network IDS Architecture

This system is a modular network-based intrusion detection system that analyzes tcpdump data to develop windowed traffic intensity trends. Because of our learning approach, many of the components in our system need time to be trained on the traffic intensity before detection is possible. The first training phase is named the architectural learning phase, because during this time, we select machine ports for monitoring, therefore determining the structure of the MLP neural network. The second training phase involves training the SOM to be able to cluster windows of traffic behavior by activity. Lastly, the third and final training phase involves training the MLP network for traffic classification.

The system flow is depicted in Figure 3.8. The system reads in tcpdump data and sends it first to the preprocessing module which keeps statistics of the traffic intensity on a source by source basis in each time interval. At any given point, there can be many sources in communication with the victim. Therefore, simply

grouping the information by sources will not create a uniform representation of data for the MLP network. To address this issue, we send the preprocessed source information, which contains the traffic intensity from a source to the target machine, to a SOM clustering module which groups similar traffic intensity trends together into behavioral clusters. This type of behavior-based clustering is desired because a machine may be used as an attack source in one time interval and as a normal source in the next time interval. Once the traffic intensity is grouped into clusters, the group statistics are then normalized and sent to the MLP neural network to render a decision as to the likelihood of a pending attack.

Over time, the dynamics of the traffic seen by our system are bound to change. To account for these changes our system can be retrained. To provide a smooth process for this activity we can pipeline the detection and training efforts. This simply means that while our detection components are running, we will also engage our learning modules to learn on the new traffic behavior. The new learned behavior would be instituted into the detection components when available.

3.3.2 Data Collection and Preprocessing

We use *tcpdump* to collect the headers of packets traversing a network. Some fields of interest contained in headers are listed in Table 3.5. The DARPA evaluation

Field	**Definition**
Timestamp	Time packet was received by tcpdump
Source	Machine that is sending packet
Destination	Machine receiving packet
Protocol	Protocol used to transmit packet
Source Ports	Service from which packet is sent
Destination Port	Service to which the packet is sent

Table 3.5: Packet header fields

program has indexed multiple traces of network attacks in the data we used. Once the data are collected, it is sent for preprocessing to prepare for the cluster and MLP training modules.

The preprocessing module formats the tcpdump data for the learning phases of our system. The fields from the packet header that are important for our system

are the timestamp, source, destination, and destination port fields. The network data are processed in time-windowed intervals Δt on a source by source basis. In other words, all of the traffic seen going to a particular host in a particular time segment Δt is grouped together in the preprocessing stage. The preprocessing module characterizes the activity observed in each Δt by keeping a count of the number of times a particular source contacts a particular port on the victim machine.

3.3.3 The Architectural Learning Phase

As previously stated, the preprocessing module characterizes the activity observed in each Δt by keeping a count of the number of times a particular source contacts a particular port on the victim machine. However, to allow a machine to differentiate one application's traffic from another, hosts have up to 2^{16} or 65,536 ports through which the traffic may travel to and from [151]. It can be difficult for an intrusion detection system to monitor all ports to determine if an attack is occurring. Additionally, it is unlikely that all of the ports on a machine are active at the same time. More importantly, we must have a consistent structure for representing the data to our neural network. In this sense, the number of ports we actually choose to monitor will dictate the architecture of our neural network. Therefore, we establish the M ports that are important for us to monitor in the "*architectural learning phase*", making our representation of activity in a particular Δt a M-element array per source.

Recall that in the preprocessing module, network traffic is monitored, keeping track of the number of times host ports are accessed in a time interval Δt. In the architectural learning phase, preprocessed data from several Δt intervals are monitored to discover the most active ports on the host. The actual duration of the learning phase is determined by an architectural multiplier F (simply a coefficient), which is multiplied by the time interval Δt. The network traffic is observed for a period of time equal to $F * \Delta t$, cataloging the number of times sources access the different ports on a target machine. These port accesses form the set A. In the beginning, the system administrator defines a list of known ports to monitor KP and the number of extra ports ep that the system can add to KP. At the end of the

architectural learning phase, the most active ep ports not already in KP are added to the set KP creating the $FINALSET$. The formula for $FINALSET$ is shown in Equation 3.1.

$$FINALSET = KP \cup \max(ep, A) \tag{3.1}$$

An example of this process is shown in Figure 3.9, where the system administrator requested the addition of two ports to create $FINALSET$ (i.e., $ep = 2$).

Ports Given

21
22
23

Extra Ports

ep_1
ep_2

+

Activity during F * Δt

Ports	Hits
21	40
20	17
22	17
23	15
25	24
80	49

=

FINALSET

21
22
23
25
80

Figure 3.9: Illustration of combining given ports with activity seen during the architectural learning phase to find the final set of ports

After the architectural learning phase, only those ports comprising the $FINALSET$ are monitored and presented to the remaining parts of the system. An example of the preprocessed data before the architectural learning phase can be found in Table 3.6. An illustration of the same data after the architectural learning phase can be found in Table 3.7.

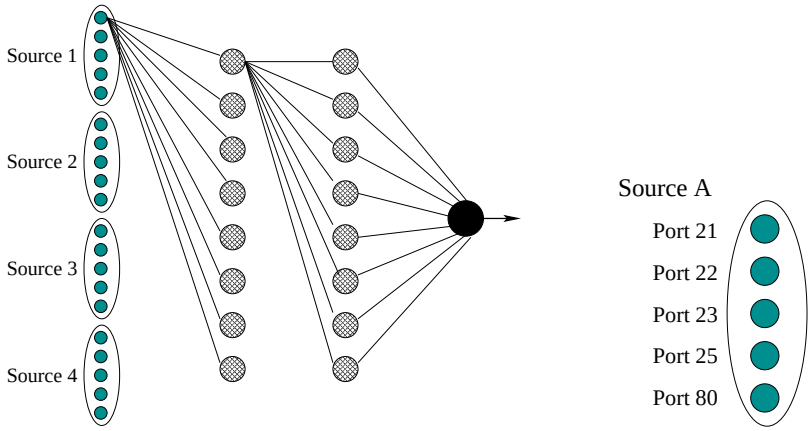
Once we have determined the ports to monitor, the neural network structure can be established as having $N * M$ input nodes in which N is the number of sources and M is the number of monitored ports ($|FINALSET|$). The first M nodes of the neural network input layer represent the total number of packets sent from the first source to the corresponding monitored port. The next M nodes of the input layer receive the respective total numbers of packets for the second source in the same order as the first layer, and so forth. An example of this architecture for four sources is given in Figure 3.10(a and b). In this example, the ports are the same as those selected to be in the $FINALSET$ in Table 3.9. If only three sources have had communicated with the target machine, then the architecture would have contained three sets of M nodes in the input layer.

	Time	Source	Dest. Port
first Δt	92635327	10.1.9.2	80
	92635328	10.1.9.2	20
	92635328	10.1.9.5	23
	92635331	10.1.6.2	21
	92635334	10.1.9.5	161
	92635338	10.1.6.2	21
	92635342	10.1.6.2	80
	92635344	10.1.9.2	21
	92635349	10.1.9.5	23
	92635354	10.1.9.5	21
Δt *Separator*			
second Δt	92635360	10.1.9.2	80
	92635360	10.1.9.5	20
	92635361	10.1.6.2	80
	92635362	10.1.6.2	21
	92635363	10.1.9.2	161
	92635365	10.1.9.5	21
	92635371	10.1.9.5	23
	92635373	10.1.6.2	80
	92635373	10.1.9.2	21
	92635375	10.1.6.2	25
	92635379	10.1.9.2	21
Δt *Separator*			

Table 3.6: Preprocessed data before the architectural learning phase.

	Source	Port 21	Port 22	Port 23	Port 25	Port 80
first Δt	10.1.9.2	1	0	0	0	1
	10.1.9.5	1	0	2	0	0
	10.1.6.2	2	0	0	0	1
Δt *Separator*						
second Δt	10.1.9.2	2	0	0	0	1
	10.1.9.5	1	0	1	0	0
	10.1.6.2	1	0	0	1	2
Δt *Separator*						

Table 3.7: Table 3.6's data grouped into port numbers obtained by the architectural learning phase (Figure 3.9).



(a) NN structure with 4 sources, each with 5 inputs.

(b) Source input view.

Figure 3.10: Neural Network Structure

As mentioned earlier, we use a neural network, whose structure must be consistent, to analyze the traffic activity and recognize anomalous behavior. As pictured in Figure 3.10, the input to our neural network detector is the number of connections per port made from each active source to a monitored machine during a single time interval. Computing the $FINALSET$ has fixed the number of ports we will monitor and present to the neural network. However, the monitored machine could receive connections from any number of sources in a particular Δt. The number of sources presented to the neural network, much like the number of ports, must also be fixed. To account for all of the sources that may be active, we cluster the different sources into a consistent number of behavioral groups (N). Each cluster contains any number of sources with similar network activity (including no sources at all). Statistics from each of the N behavioral groups are presented to the neural network at every time interval. To accomplish this clustering, we use another neural-based technology, Self-Organizing Maps (SOMs), with a special automated clustering algorithm.

3.3.4 Clustering with Self-Organizing Maps

Clustering network traffic has been shown to be an effective way of classifying similar trends [129, 125]. To provide both a visual representation of the traffic trends as well as a meaningful clustering technique, we use a SOM. SOMs, unsupervised learning tools, are most known for their visual clustering abilities, however we have developed a frequency-based automated clustering technique for the SOM to use in our DoS detector. The details of the frequency-based automated clustering algorithm can be found in Appendix D, while its application to this problem is explained here.

3.3.4.1 The SOM Training Phase

Our SOM contains a grid of neurons each holding a weight vector of size $|FINALSET|$ which evolves to represent the input values from the collected data elements.

Initializing the SOM neurons can be done in many ways. The traditional initialization procedure is to simply choose random values for initial weight vectors. However, when we used this method, only a very small fraction of the neurons in the SOM were chosen as the Best Matching Unit. This situation creates slightly malformed clusters using the frequency based clustering method (See Appendix D). Another method used by some is to initially assign the weight vectors to a random input vector [75].

Our method of initialization combines both of the aforementioned methods. We first define sets of ranges for each of the weights and observe how often input values fall in these ranges. The ranges selected should end at logical partition points containing few input values. For example, if Table 3.8 describes the data before clustering, we could look at an inverted histogram to find partition points for the dataset (Figure 3.11).

Choosing the perfect partition point or perfect number of statistical groups of the dataset is not neccessary while initializing the SOM, but the partition point should logically divide the dataset into groups that would generally appear separate in an inverted histogram. In Figure 3.11, we separate the dataset into three groups

	Source	Port 21	Port 22	Port 23	Port 25	Port 80
first Δt	10.1.9.2	8	0	50	11	220
	10.1.9.5	54	3	63	132	687
	10.1.6.2	0	37	0	0	0
	10.1.6.5	0	0	13	0	54
Δt Separator						
second Δt	10.1.9.2	0	0	120	0	0
	10.1.5.3	154	23	0	0	0
	10.3.6.5	0	0	60	0	286
	10.3.9.8	76	0	0	187	0
	10.3.9.17	0	0	73	0	321
	10.4.16.64	0	112	0	0	0
Δt Separator						

Table 3.8: Example of preprocessed totals before clustering

using the partition points 15 and 80.

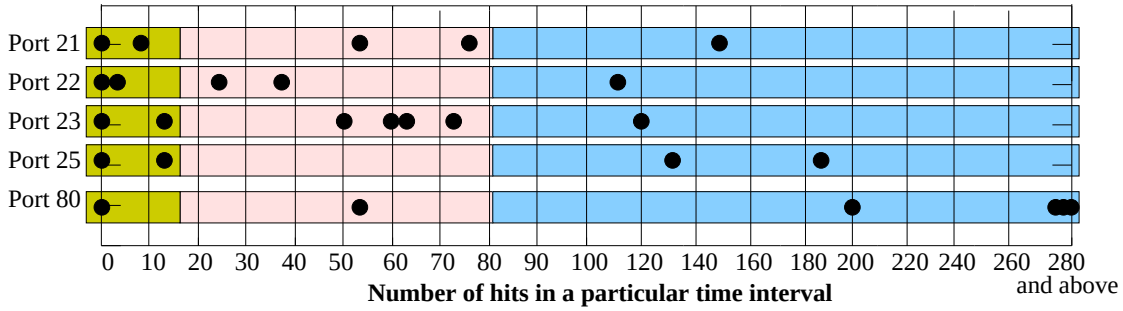


Figure 3.11: Graph of data ranges

Table 3.9 reflects the statistical distribution of Table 3.8's data where the ranges are $[h < 15]$, $[15 \leq h \leq 80]$, and $[80 < h]$.

3.3.4.2 SOM Clustering Expert

When the SOM has been trained, an input vector is presented to the SOM, and the SOM returns which behavioral cluster the input belongs according to the frequency-based clustering algorithm (Appendix D). Behavior from sources clustered together is statistically combined (either summed or averaged). For example, if Table 3.10 refers to the behavioral cluster organized by the SOM, Table 3.11 would reflect the sum of all source behavior in the same cluster.

Port 21			···	**Port 80**		
$h < 15$	$15 \leq h \leq 80$	$80 < h$	···	$h < 15$	$15 \leq h \leq 80$	$80 < h$
7	2	1	···	5	1	4
Percentages						
70%	20%	10%	···	50%	10%	40%

Table 3.9: Distribution calculations from totals listed in Table 3.8, used for SOM initialization. In this table, h is the total hit count in a particular range of the specified port.

	Cluster #	**Source**
first Δt	Cluster 1	10.1.9.2
	Cluster 2	10.1.9.5
	Cluster 3	
	Cluster 4	10.1.6.2
		10.1.6.5
Δt Separator		
second Δt	Cluster 1	10.1.9.2
		10.1.5.3
	Cluster 2	10.3.6.5
		10.3.9.8
	Cluster 3	10.3.9.17
	Cluster 4	10.4.16.64
Δt Separator		

Table 3.10: Example of clustered sources decided by the SOM using data from Table 3.8

3.3.5 Decision Making with the MLP Network

Now that the input is consistent in the number of clusters and ports monitored per cluster, the MLP network can be created. Because there is not an exact science to determine all of the architectural configuration of the neural network (e.g. number of hidden layers, number of hidden nodes in each hidden layer), we wrote scripts to run and test many different configurations. We always specified 4 ports, but varied the additional ports the application should add between 1 and 2. We created networks with up to two hidden layers, each having from $0.8 * number\ of\ input\ nodes$ to $2 * number\ of\ input\ nodes$ as the number of hidden nodes in each layer (please

	Cluster #	Port 21	Port 22	Port 23	Port 25	Port 80
first Δt	Cluster 1	8	0	50	11	220
	Cluster 2	54	3	63	132	687
	Cluster 3	0	0	0	0	0
	Cluster 4	0	37	13	0	54
Δt Separator						
second Δt	Cluster 1	154	23	120	0	0
	Cluster 2	76	0	60	187	286
	Cluster 3	0	0	73	0	321
	Cluster 4	0	112	0	0	0
Δt Separator						

Table 3.11: Example of data from Table 3.8 clustered into clusters discovered in Table 3.10

note, the *number of input nodes* will change with the number of ports monitored by the application). The neural network with the lowest number of nodes and best error rate is described below:

- We specified 4 specific ports and 1 extra port to be chosen by the application.
- The MLP network we used had 4 behavioral clusters (each with 5 ports), creating an input layer of 20 neurons.
- There was one hidden layer containing 25 neurons, and an output layer containing a single neuron (See Figure 3.12).

The neural network was trained using a back-propagation algorithm with early stopping and a maximum of 10,000 training epochs. It was trained using 100 data samples, and tested on 50 samples. Care was taken in preparing the training sets to not bias the neural network towards recognizing certain cases (i.e. there were as many attack cases as there were normal cases).

3.3.6 NN Structure Provider

Our system is sensitive to the learned structures of the SOM and the weight structure of the MLP network. It is certainly reasonable that at some point the behavior may change and the learned structures may no longer be as useful. The

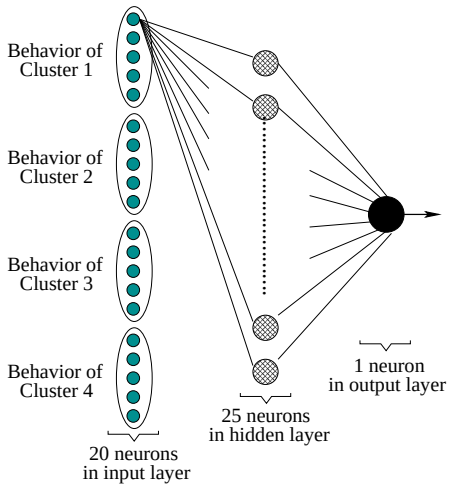


Figure 3.12: Clustering Neural Network Structure

NN Structure Provider facilitates the updating of both the SOM and MLP weight structures by providing new structures to the SOM clustering unit and the MLP traffic classifier without interruption to the rest of the system. When the SSO or an offline training program wants to replace these structures, it simply interacts with the NN structure provider and gives it the new structures which will be forwarded to the appropriate components. The relationship of the NN structure provider to the rest of the system can be seen in Figure 3.8. As will be described with more detail in Section 7.2.2, this component makes it possible to develop a distributed modular architecture permitting a more automated approach to updating traffic signatures.

3.3.7 Test Results

As pictured in Figure 3.8, our system is built to run on either historical tcpdump binaries or from a real-time tcpdump process. For testing purposes, we ran our system using tcpdump binaries from the DARPA 1999 training dataset [109, 110]. Once again, in our testing, the Δt value used was 40 seconds and the neural network is trained using 100 training patterns and 50 testing patterns equally balanced with attack and non-attack traffic.

We had a great deal of trouble trying to get our neural network to detect all types of attacks simultaneously. However, the neural network performed well when tested on individual types of attacks, one at a time. Our training and testing in

this area was limited however, because our dataset did not contain many instances of the same attack. Table 3.12 shows some of our results where *sshprocesstable* is the name of a particular type of denial of service attack. In Table 3.12, columns 2, 3, 4, and 5 respectively refer to the correct prediction of normal traffic trends, the incorrect prediction of normal traffic trends, the correct prediction of attack traffic trends, and the incorrect prediction of attack traffic trends. Currently, the *sshprocesstable* attack was the only attack for which we had enough data to mix into both the training and test set of the MLP network.

	Correct Normal Predictions	False Negatives	Correct Attack Predictions	False Positives
Union of All Attacks	100%	0%	24%	76%
Sshprocesstable	100%	0%	100%	0%

Table 3.12: Current detection results

3.3.8 Conclusion

Many methods have been employed for intrusion detection. However, modeling networking trends for a simple representation to a neural network shows great promise, especially on an individual attack basis. Also, using SOMs as a clustering method for MLP neural networks is an efficient way of creating uniform, grouped input for detection when a dynamic number of inputs are present. Once trained, the neural network can make decisions quickly, facilitating real-time detection. In this study we have found the neural networks using both supervised and unsupervised learning have many advantages in analyzing network traffic trends.

3.4 Related Works

Due to the increasing complexity of computer networks as well as the growing sophistication of network-based attacks, network-based intrusion detection has recently gained more attention from academic, military, and commercial sectors. Consequently, various IDSs have been implemented that address different aspects of network security and use different methods of detection. In this section, we briefly discuss some of the general intrusion detection research efforts. We then discuss spe-

cific research efforts as they relate to state, neural networks, and clustering based methods. In depth reviews of these systems and more can be found in [9, 6].

A group at MIT designed several applications to address different aspects of intrusion detection. The first was a password guessing detector which scans telnet connections containing only failed logins. It identifies password guessing attacks based on the number of connections between host pair and the user-names/passwords tried [179]. The authors created the password guessing detector to detect "doorknob rattling" and "dictionary" attacks. Doorknob rattling involves attempting to access a host using default or incorrectly configured accounts that have not been removed. Examples of this include looking for a "guest" account which will allow anyone to log on, but with reduced permissions, or a "root" account with no password. A dictionary attack simply involves a large-scaled attempt to log into user accounts with passwords very similar to the user name or dictionary words. This system is also a misuse detector, trained to detect the password guessing behavior of those executing the dictionary and doorknob rattling attacks.

NSM (Network Security Monitor) is an expert system based tool that analyzes data for attacks and suspicious behavior [76]. The system has a set of identified keywords for which it keeps current counts. The counts are then combined to come up with an overall danger score. If the score is past a certain threshold, an alarm is caused. This is a misuse detector that is trained on intrusion data.

GrIDS (Graph-Based Intrusion Detection System), constructs activity graphs from network traffic data to detect large-scale automated attacks in real-time [148]. GrIDS uses a graphical representation to monitor the activity of not just a user, or a system, but an entire network.

Several research groups use deception to thwart network-based attacks. In these cases, functional and procedural components are setup to divert the activity of a potential intruder from real, valued assets to false assets. These efforts are not only used to detect intrusions, but they are also helpful in gathering information about the attacker. Real users have no reason to use these false assets, so any access of this type is an immediate indication of an intruder. Once the false asset or machine has been accessed, the administrators can closely monitor the attacker

to learn their attack methods, motives, etc. An overview of different deception-based intrusion detection groups and their techniques can be found in [6, 35]. Some corporate groups have also used this technique (e.g. Symantec™ has a deception-based intrusion detection product called ManTrap® [136]).

3.4.1 State Based Intrusion Detection Methods

Our TDFA IDS shares methodologies similar to those of a couple of other IDSs. For instance, Snort is a lightweight network-based IDS that utilizes a rule-based approach, along with network packet-sniffing and logging to perform content pattern matching and detect a variety of attacks [137]. In comparison, we also use a rule-based approach in detecting attacks, but we apply it in the form of deterministic finite automata as opposed to the rule chains used in Snort.

Another tool comparable to our TDFA IDS is NetSTAT. It is an extension of the original STAT design which models an attack as a sequence of actions that progressively take a computer from an initial normal state to a compromised state [85, 174]. NetSTAT applies that model to a networked environment by modeling both the guarded network and the attacks. Based on this model, it determines which network events have to be monitored and where in the network they should be monitored. We choose to bypass modeling the network and focus directly on the representation of various DoS attacks. Overall, this choice yields a more compact system and yet does not affect detection accuracy (as demonstrated by our results presented in Section 3.2.4). Our method also extends NetSTAT's model in one additional aspect; we consider the time intervals between system events when defining the state transitions of the attack signature.

3.4.2 Neural Networks for Intrusion Detection

There are a few different groups advocating various approaches to using neural networks for intrusion detection. A couple of groups created keyword count based misuse detection systems with neural networks [111, 138]. The data that they presented to the neural network consist of attack-specific keyword counts in network traffic. In these methods, the network data is grouped into user sessions and the user-entered commands are then filtered. In a different approach, researchers

created a neural network to analyze program behavior profiles instead of user behavior profiles [65]. This method identifies the normal system behavior of certain programs, and compares it to the current system behavior. Cannady developed a network-based neural network detection system in which packet-level network data was retrieved from a database and then classified according to nine packet characteristics and presented to a neural network [25]. This method is different from ours because Cannady proposed detection on a packet level, whereas we use a time-window method. Our method allows us to generalize input beyond Cannady's method, enabling us to recognize longer multi-packet attacks.

To detect other attacks in telnet sessions, a neural network-based keyword weighting system is used. This system, similar to the strategy mentioned in [111], counts the occurrence of forty keywords and uses these values as inputs to a neural network. The neural network will then determine if these statistics represent an attack [179].

One system using a key string selection algorithm and neural networks is used to detect and label User-to-Root attacks [109, 110]. The key string algorithm basically pulls important string values (usually specific commands typed by the user) out of network sniffed data (similar to the method mentioned in [76]). The strategy involves recreating sessions based on network sniffed data, and counting the occurrences of a selected set of keywords. The authors of this work keep a count for 30 keywords (i.e. "root:", "permission denied", "showmount", etc.). Every time one of the keyword sequences is seen in the data, the count for that keyword sequence is increased. These keyword counts are used as inputs to a neural network to determine if an attack is taking place in that session. This approach could be considered a misuse detector as the neural network is trained to recognize the intrusion behavior based on the number of occurrences of certain keywords. Another neural network is used to label the attack.

3.4.3 Clustering for Network Traffic

There are also a few groups advocating various clustering approaches for anomaly based intrusion detection. Self-Organizing Maps have been used as anomaly

intrusion detectors [67] by clustering system logs into two dimensional space for graphical visualization. SOMs have been used to cluster and then graphically display many types of data, enabling a user to determine the data's classification through visual inspection. K-means clustering has also been used to aggregate the network activity of a group of machines to determine if this activity is normal or unusual [113].

CHAPTER 4
Host-Based Intrusion Detection

While firewalls, strong authentication, and network-based intrusion detection methods help to secure systems from unauthorized access and thwart attacks to network services from the outside, they do little to protect against hackers who have already gained access or authorized users with malicious intent. Studies have shown that, in many cases, a companies biggest security threat comes from authorized users in the organization [118]. Host-based intrusion detection fills many of the voids of other systems in detecting this type of activity. Host-based intrusion detection involves monitoring various system audit logs for indication of suspicious behavior on a particular host. These logs are traditionally created and maintained by processes individually installed and run on each host machine. A table of logs used in different host-based intrusion detection efforts is included in Table 4.1.

Log Files	
system logs	file checksum logs
event logs	process accounting logs
network logs	user audit logs
session logs	disk usage logs
security logs	

Table 4.1: Table of log files used in host-based intrusion detection

Some host-based detection efforts also examine the network traffic to and from a particular host in hopes of detecting suspicious activity. These processes are separated from network-based intrusion detection because they only monitor data to and from the machine from which they are running.

Host-based intrusion detection is not a new concept. Many break-ins have been contained by attentive system administrators who have noticed something "different" about their machines or users. As computers and networks continue to grow to be a more integral part of the way we do business and communicate, system administrators will not be able to manually monitor all of the computer systems.

In addition, attacks have become more sophisticated and more frequent, rendering even the most attentive system administrator helpless in monitoring his or her many systems without automated processes.

Host-based detection systems can detect attacks that network-based detection cannot see. For example, attacks that are made at the keyboard of a critical machine do not travel across the network. More information, including machine and process state, are available for host-based methods enabling coordination of multiple data sources in detection. Because host-based systems are installed on each machine, encrypted and highly segmented or switched environments do not affect the deployment of the system (as it does network-based systems).

In our work, we focus on detecting masquerading attacks. These are attacks in which an intruder gains access to another user's account. Once access is gained, the intruder will normally provide a way for repeated access. These are particularly dangerous attacks because if a user with high privileges is compromised, sensitive information and vital business components could be exposed. These attacks can also last for a very long period of time if gone undetected.

4.1 Current Standards

Current host-based intrusion detection systems use kernel logs, audit data, file tracking systems, and other proprietary monitoring tools to analyze the host system for signs of misuse. Some systems also use network monitoring tools to look at the packets coming in and out of the host. These are still considered host-based systems because they are analyzing the traffic for only one host.

Host wrappers or personal firewalls can be configured to look at all network packets, connection attempts, or login attempts to the monitored machine. This can also include dial-in attempts or other non-network related communication ports. Examples of well-known wrapper packages can be found in [189].

Most host-based agents monitor access, changes in user privilege, and user activity. Some well-known commercial versions which do such capabilities include products from Symantec [158], NFR Securities [123], and ISS (Internet Security Systems) [87, 86]. These systems typically monitor the system for well know exploits,

and are sometimes coupled with network-based intrusion detection systems. Some host-based systems, such as Tripwire [97], specialize in monitoring the state of important files on the host, both throwing an alarm when this state has changed and keeping track of the last stable state.

Many of the existing systems are not capable of detecting a masquerading host-based attack until the masquerader tries to cause damage to the system through well know exploits. However, a masquerader wanting to preserve his access to a given system will intentionally attempt to attract little attention to himself, possibly going undetected forever.

4.2 Probabilistic State Finite Automata Host-Based Intrusion Detection

We have created a host-based intrusion detection network manager particularly aimed towards detecting masquerading attacks. Ours is an anomaly detection system (recall the difference between anomaly systems and misuse systems described in Section 1.1.2). We first generate a signature of normal behavior for each user of a computer system. We make the assumption that each user has a sequence of commands that he or she frequently uses. We extend this assumption by assuming that we can characterize a user's behavior by these frequently used sequences of commands. The user might search for a given directory, open a text editor, check his or her mail, compile a program, etc. By having a signature of all the frequent (and infrequent) command traces that a user types, we can compare future command traces that the user will type against the signature. Because we store the actual command traces, we not only have a representation of the frequent commands the user types, we also have a representation of the orderings between commands.

Anomalous behavior is defined as any behavior that sufficiently deviates from the model. Thus, the anomalies that this tool detects may or may not correspond to an actual intrusion. In some instances, the user may simply be experimenting with a new set of commands, or the user's behavior may be different due to fatigue or stress. Other times, there is in fact another person masquerading as the user. In either instance, the tool described in this section may detect that the shown

behavior deviates from the model.

4.2.1 What is a Probabilistic State Finite Automata?

A probabilistic state finite automata (PSFA) can be viewed as an extension of a Finite State Machine. It has the general structure of a Finite Automata, but attaches a probabilistic value to each state. An example of a PSFA can be found in Figure 4.1. The probabilistic value associated with each node, $P(m)$, is the

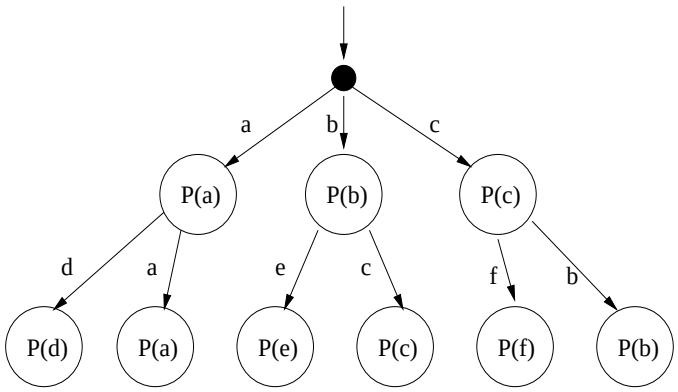


Figure 4.1: PSFA example

probability of reaching the particular node from the previous node. This probability can be computed from a sample population while building the PSFA.

4.2.2 System Overview

As mentioned in Section 4.2, the tool described in this chapter generates signatures of each user to be used when testing future command traces. The system consists of four basic modules (shown in Figure 4.2). The first module, the Filter Module, is responsible for extracting user commands from log files, then converting the user commands with the associated timestamps into command traces. These command traces are then either passed to the PSFA Builder which constructs the PSFA, or to the PSFA Tester for testing of future command traces. The PSFA Provider is responsible for passing the appropriate PSFA to the PSFA Tester.

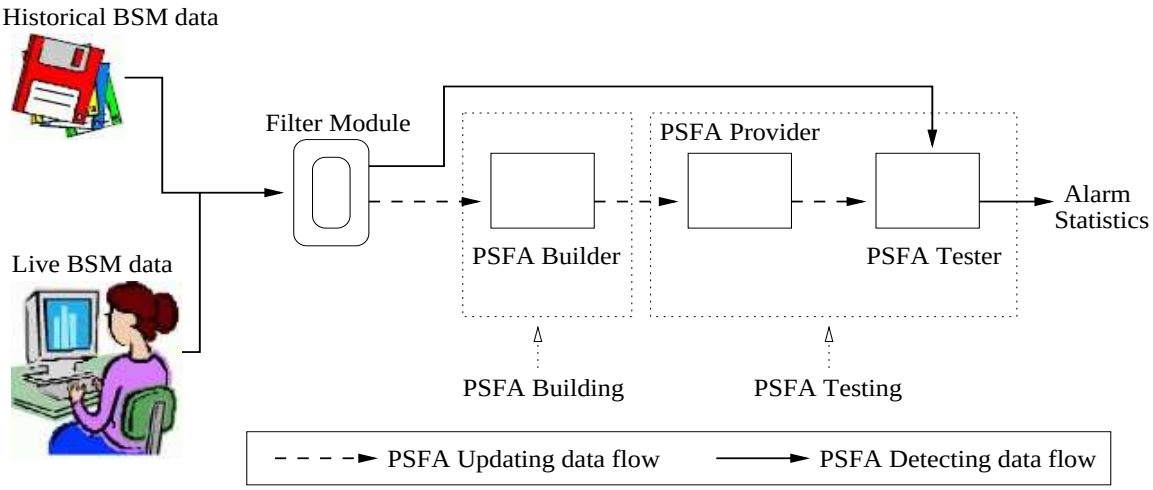


Figure 4.2: PSFA System Architecture

4.2.3 Data Collection

To detect attacks within a host, we have studied techniques that use the well-known Solaris BSM (Basic Security Model) [155] audit data and the UNIX *acct* accounting mechanism. BSM reports user sessions at the system call level while *acct* does the same at the user-typed command level. Our system can handle input from both of these processes.

4.2.4 Filter Module

The responsibility of the filter module is to parse through the audit data and produce a string of user commands and timestamps which can be used by the PSFA builder or the PSFA tester to build or traverse the automata, respectively.

BSM stores its audit data in binary format which must first be converted into an ASCII format before processing. Once converted, the ASCII file will reflect all of the system calls created by the many commands that a user may type or initiate through a graphical interface. For every action of the user, like opening a file, typing a command, moving the mouse, etc., many system calls are generated. We pay particular attention to the *execve* command which is the system call used by the shell to execute most user-entered commands. Some system calls that also reflect used-entered commands but are not captured by the *execve* call include *login* and

chdir. These additional calls are also parsed into the string of commands created as output from the filter module. The Unix *acct* data is already in ASCII format and simply needs to be slightly reformatted into the string format used in the following modules.

The string format exported from this module simply contains the timestamp followed by the name of the command as shown in Figure 4.3.

$$Trace_{user} = < ts_1 > \sqcup < c_1 >, \sqcup < ts_2 > \sqcup < c_2 >, \sqcup \cdots < ts_n > \sqcup < c_n >$$

where: $\sqcup$ = space character
ts_t = timestamp at time t
c_t = command entered at ts_t

Figure 4.3: String representation of a user's command trace

This process is repeated for each log file used for training. It is essential that the log files we use to construct our signatures are created from commands that the user, and not a masquerader, typed. Otherwise, we would be modeling signatures of the attacker and most likely label the attack behavior as normal.

4.2.5 PSFA Builder

Once we have the listing of commands and timestamps we can convert them into a series of command traces. By modeling the command traces entered by the user rather than just the individual commands, we develop a more accurate representation of user behavior. The command traces that are generated are completely dependent on the time intervals between successive commands. In order for two successive commands to be considered in the same command trace, the difference between their timestamps must be less than the command inter-arrival threshold, δt. Figure 4.4 shows a user signature containing q commands.

A large δt will produce longer command traces, because there is a longer time window in which two successive commands can still be a part of the same trace. A very large δt could in effect store an entire user session as one command trace. A

$$Signature_{user} = < ts_1 > \sqcup < c_1 >, \sqcup < ts_2 > \sqcup < c_2 >, \sqcup \cdots < ts_q > \sqcup < c_q >$$

where: $\{\forall i \in 2, 3, .., q\} | ts_i - ts_{i-1} \leq \delta t$
$\sqcup$ = space character
ts_t = timestamp at time t
c_t = command entered at ts_t
δt = command inter-arrival threshhold

Figure 4.4: String representation of a user signature

small δt, on the other hand, will produce shorter command traces. The value of δt is very important in determining the structure of the PSFA. See Table 4.2 for an example of two sets of command traces derived from the same command listing, based on different command inter-arrival thresholds.

Time	Command
10:00	login
10:01	cd
10:02	vi
10:04	ls
10:05	pico
10:06	mv
10:08	ls
10:09	mail
10:10	exit

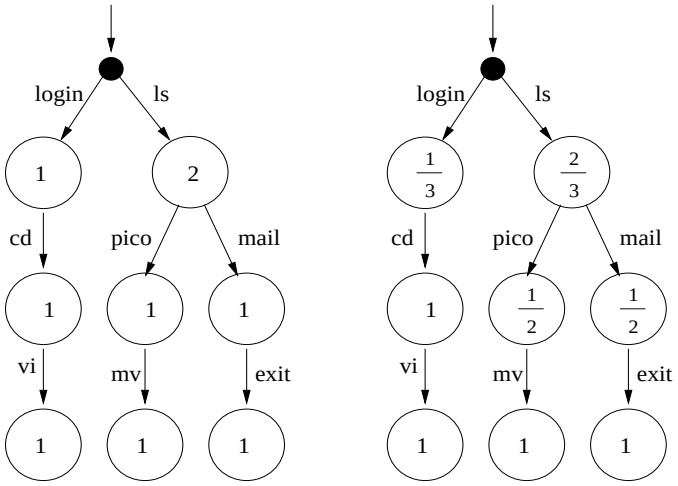
δt	Command Window
δt=1	[login, cd, vi] [ls, pico, mv] [ls, mail, exit]
δt=3	[login, cd, vi, ls, pico, mv, ls, mail, exit]

Table 4.2: Sample host command dataset

4.2.5.1 Building the PSFA From Command Traces

The PSFA for each user is a collection of all the command traces extracted from his or her log files. The PSFA is represented as a tree structure. Each node in the PSFA contains statistics as to the likelihood of reaching that node, and the transitions between nodes are used to represent how one command follows another. Because the distribution of each user command is not known, we must build our PSFA for each user from examining a sample dataset. Command traces are added to the PSFA one after another. Figure 4.5 shows the formation of a PSFA consisting of

the command traces from the Table 4.2 where $\delta t = 1$. To calculate the probability $P(m)$ during the training stage, we first keep a simple count of the number of times we reach each sequence in the PSFA (Figure 4.5(a)). When $P(m)$ is needed, we divide the number of times the node was reached from the parent node by the number of times all nodes of the parent were reached. Using the same example PSFA from Figure 4.5(a), we would compute the probability $P(m)$ of reaching the node containing the 'ls' transition by dividing the number of times 'ls' is reached from its parent, 2, by times reached from all nodes reachable by its parent, 3. Thus the probability of the 'ls' transition would be $\frac{2}{3}$. Figure 4.5(b) shows the new automata in which all the nodes have computed $P(m)$ values.



(a) Sample PSFA totals created from sample host data shown in Table 4.2.

(b) The PSFA with probabilities in the nodes of the automata.

Figure 4.5: Formation of PSFA from sample data in Table 4.2 where $\delta t = 1$.

At this point each node in the automata has a probability $P(m)$ associated with it, corresponding to the probability it will be reached from its parent. We can now trace down individual command traces to compute the probability of the command trace. For each command trace, we compute the multiplied probability P_m to be the product of the probabilities $P(m)$ of each node we encounter as we

traverse the PSFA. We also calculate the average probability P_a from all of the $P(m)$ values associated with each node we encounter. From Figure 4.5(b), if we traced down the command trace $ls \rightarrow pico \rightarrow mv$, we would compute P_m and P_a for the given command trace as done in Equation 4.1.

$$\begin{aligned} P_m = & \ \tfrac{2}{3} * \tfrac{1}{2} * 1 = \ 0.333 \\ P_a = & \ \tfrac{\frac{2}{3}+\frac{1}{2}+1}{3} = \ 0.722 \end{aligned} \tag{4.1}$$

With the ability to trace down the automata and compute P_m and P_a for each command trace stored in the automata, we now can distinguish frequently seen command traces from infrequent command traces. At this point we are able to test future command traces against the automata using P_m and P_a. If P_m and P_a are significantly low (below a predefined threshold), the command trace being tested is determined to be anomalous.

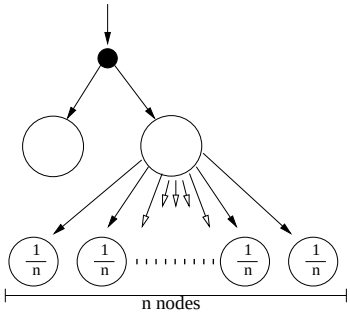
4.2.5.2 Updating the Probabilities Using Standard Deviation

Strictly computing the multiplicative probability P_m and the average probability P_a introduces a unintentional bias against two cases of advanced users.

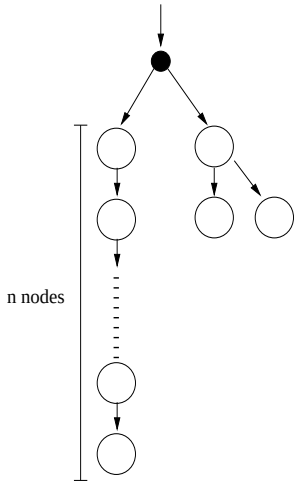
- **Case 1:** The first case involves users who frequently use a large variety of commands. If at any point, the next step in a user signature can be 1 of any n commands, when n grows to be large, the probability of each individual command grows smaller. For example, Figure 4.6(a) illustrates a PSFA where a user uses a large number of commands with a uniform distribution for each. In this case, as n grows to infinity, the probability of each command ($P(m)$) drops to 0. There is a harsh penalty imposed on the user, no matter which command they use. This case affects both the multiplicative and average probabilities.

- **Case 2:** A similar penalty is present when a user types commands quickly. A very quick typist could have very long signature traces because all of his or her commands would be entered within δt of each other. The long signature traces

has the worst effect on the multiplicative probability (P_m) because many of the probabilities to be multiplied will be less that 1. Figure 4.6(b) describes a PSFA with a particularly long signature trace in it. This signature is composed of n commands, each containing some $P(m)$ where $0 < P(m) \leq 1$. As we traverse this particular trace multiplying P_m by n values less than or equal to 1, P_m can drop to near 0.



(a) PSFA of a user who uses a large number of commands



(b) PSFA of a user who enters commands very quickly

Figure 4.6: Special PSFA cases of advanced system users

To address these special cases, we use a standard deviation approach. In the first case, we compute a new standard deviation based statistic, $P_\sigma(m)$, for each node using the individual node probabilities, $P(m)$. Essentially, $P_\sigma(m)$ is based on how many standard deviations the probability of reaching the node is from the average probability of reaching all the sibling nodes. Its exact formula can be found in Equation 4.2.

$$P_\sigma(m) = \begin{cases} 1 + C_{reward} * \frac{P(m) - P_{avg}}{\sigma} & if\ P(m) > P_{avg}(m) \\ 1 + C_{penalty} * \frac{P(m) - P_{avg}}{\sigma} & if\ P(m) < P_{avg}(m) \\ 1 & otherwise \end{cases} \tag{4.2}$$

$P(m)$	the probability for a particular state
$P_\sigma(m)$	standard deviation-based anomaly metrics for a particular state
C_{reward}	reward coefficient
$C_{penalty}$	penalty coefficient
$P_{avg}(m)$	the average breadth probability found in the level and branch of $P(m)$
σ	the standard deviation found in the level and branch of $P(m)$

Permitting values for P_σ which are greater than 1, allows us to reward future command traces that reach that particular state.

In the second case, where the length of the trace is very long, we use a standard deviation approach based on the average length of traces in the PSFA. However, in this case we simply compute an additional reward when following a particular trace that is longer than the average trace size. This reward is based on how many standard deviations away the current trace's length is from the average trace length and would be added to the corresponding P_m value. Its exact formula can be found in Equation 4.3.

$$DepthOffset(m) = C_{reward} * \frac{L(m) - L_{avg}}{\sigma_{len}} \quad if\ L(m) > L_{avg} \tag{4.3}$$

$DepthOffset(m)$	the offset used to account for fast users
$L(m)$	Length of the particular trace
C_{reward}	reward coefficient
L_{avg}	the average length of traces in the PSFA
σ_{len}	the standard deviation of trace lengths

This offset simply rewards a user for following a longer-than-average signature trace. A reward that will in most cases supplement the cost of multiplying P_m by the fraction, $P(m)$, many times.

It is important to note that each time a command trace is added to the PSFA, the probability $P(m)$ that nodes will be reached, the average probability P_{avg} of all nodes on a given level, the standard deviation-based anomaly metrics $P_\sigma(m)$ and the $DepthOffset(m)$ may all have to be recalculated.

4.2.5.3 Mutation Events

If the next command entered, while traversing the PSFA, matches none of the commands in the next level of the PSFA, a mutation event has occurred. This event is called a mutation because some strategies consider this action to be the sign of a new mutation of an existing signature. In these cases, we do not stop traversing the PSFA, we simply determine a probabilistic value for the command in question $P(MissedCommand)$ and proceed to the next command. The following are three strategies used to determine $P(MissedCommand)$:

- **Default Value** This method simply returns a default (relatively low, e.g. 0.001) probability when encountering a command currently not in the corresponding position in the corresponding trace ($P(MissedCommand) = 0.001$).

- **Inverse Usage Computation** The idea behind the inverse usage method is the more popular a particular signature is in the PSFA, the more the user should be penalized for not completing the signature or one of its recorded variations. To compute values with this philosophy, we keep track of the number of times each PSFA node was reached while training, k (see Figure 4.7). We then use the inverse of the number of times the last matching node in the automata was reached ($\frac{1}{k}$), because this usage metric identifies the number of times the user used the same partial trace, but finished it with valid commands. For example, in Figure 4.7, consider the valid user traces [cd,ls,cd,more] and [cd,ls,cd,grep]. If the current user trace is [cd,ls,cd,lprm], we will notice that the last matching node in the trace was "cd," reached 40 times ($k = 40$). In this case the value return for the missed command, lprm, would be $P(lprm) = \frac{1}{40}$.

- **Command Frequency Computation** When developing a probabilistic value for the command missing from the signature using the command frequency computation method, the goal is to make a distinction between commands which the user rarely or never uses and those the user uses frequently. Anomalous use of a frequently used command (evaluated on a per user basis) should be penalized more than a seldom used command. To do this we keep all of the commands the user uses in a table, along with the number of times the

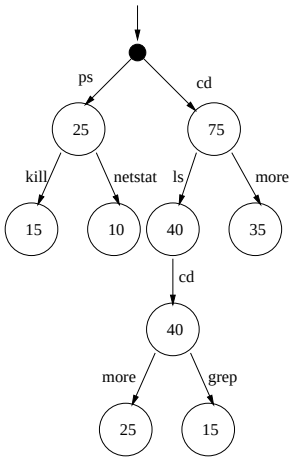


Figure 4.7: Example PSFA structure where numbers inside the nodes are frequency values

command was entered (regardless of where it is in the automata). This structure, called the Command Frequency Table (CFT), is pictured in Figure 4.8. In a mutation event, frequency of use for the missed command is obtained

Command	Frequency
cmd_1	$freq_1$
cmd_2	$freq_2$
$\vdots$	$\vdots$
cmd_{n-1}	$freq_{n-1}$
cmd_n	$freq_n$

Figure 4.8: Illustration of the Command Frequency Table (CFT)

from values in the CFT ($freq_{MissedCommand}$), and a probabilistic value is returned for the mutation-causing command according to Equation 4.4. A default value, $P(CFTdefault)$ (usually a relatively low value), is provided if the command is not found in the CFT. This value is also used in the calculation of $P(MissedCommand)$ when the command is found in the CFT as described in Equation 4.4.

$$P(MissedCommand) = P(CFTdefault) * \frac{1}{freq_{MissedCommand}} \qquad (4.4)$$

$P(CFTdefault)$ can be provided as the lowest command probability at the current level of the trace, a parameterized constant value, or some other function of the PSFA.

4.2.6 PSFA Tester

Once a PSFA is created for a specific user, future command traces can be tested against the PSFA to determine if the behavior is anomalous or not. The command traces to be tested against the PSFA will be generated in the same manner as command traces used to build the PSFA. During testing, the PSFA will return two probabilities for each command trace it tests, P_m and P_a (described in Section 4.2.5.1). In both cases, recall that the value being multiplied or averaged is $P_\sigma(m)$, the standard deviation based anomaly metrics. The process of traversing the PSFA for testing is done as follows:

- Get the next command from the test command trace and look for a transition containing the command in the next level of the automata. In the beginning, this would be the first command in the trace and the first level of the automata.
  - If there is an associated transition, follow the transition and keep track of the anomaly metrics $P_\sigma(m)$ at the encountered node.
  - If there is no transition for the command, check the CFT.
    - If the command is in the CFT, a probability is assigned corresponding to the general usage of the command.
    - If the command is not in the CFT, a default low value is returned.
- This process is repeated for all the commands in a given test trace.
- When all the commands have been evaluated, the multiplied and averaged anomaly metrics are returned.

4.2.7 Evaluation

To evaluate the performance of our method, we ran it on test data used by Maxion and Townsend [114]. The dataset was created by Schonlau et al [142] and has thus been named the SEA dataset.

4.2.7.1 The SEA Dataset

As described by Schonlau et al [142], the masquerade detection dataset consist of user commands captured by the UNIX *"acct"* accounting mechanism. For privacy reasons, only command name and user fields were used. Seventy users were monitored and their first 15,000 commands were recorded. For some very active users, 15,000 commands were generated in only a few days; however, other users took months to generate the necessary number of commands. The data was then organized in blocks of 100 commands, giving each user 150 blocks of data.

After recording the library of commands for each user, 50 of the 70 users were randomly chosen as intrusion targets. The remaining 20 users served as masqueraders whose data were interspersed into the data of the 50 targets. The first 50 blocks (5000 commands) of all users remained unaltered to serve as a training dataset. The next 100 blocks of intrusion target datasets were contaminated according to the following set of rules.

- A block is either completely contaminated, or not at all. There are no mixed blocks.
- If no masquerader is present, a new masquerader enters the following block with a 1% probability.
- If a masquerader is present, the same masquerader continues to be present in the following block with a probability of 80%.
- Data that correspond to different masqueraders are always separated by at least one block of uncontaminated data.
- Inserting masquerading data increases the number of commands observed. The data is truncated to 150 blocks per user in order not to give away the amount of masquerading data inserted.

While the exact values of the new masquerade and continuing masquerade probabilities were arbitrary, they reflect the author's three requirements:

1. There should be an arbitrary number of masqueraders in the data (including the possibility of none).

2. The length of the actual masquerading attack should be varied.

3. Most of the user data should not be contaminated.

4.2.7.2 Conforming to the Test Data

As mentioned earlier, the SEA dataset used captured user commands from the UNIX *acct* accounting system. From this accounting system, only the command name and user name were recorded. However, our method is built on temporal differences in the data. With no timestamps given in the SEA dataset, we had to adjust our method slightly. Instead of a command trace being comprised of all commands entered within a certain time δt of each other, we had to fix the length of the command trace to an arbitrary number of commands. To this end, we fixed all command traces to be five commands (using a non-overlapping windowing method), making each user block contain 20 command traces. Because all of the signatures were the same length, a DepthOffset was not necessary.

If timestamps were available, we would expect our technique to generate more accurate results. However, testing on the SEA dataset can still give us an idea of how our method performs and give us a basis for comparison with other methods which do not consider temporal variance in their technique.

4.2.7.3 Results

Figure 4.9 shows the results of running our method on the SEA dataset with different P_a threshold values. In this graph we can see that the hit percentage achieved by most of our methods were in the high 60th percentile (with the exception of the very high hit percentage of 83% achieved when using 0.19 as the threshold value). This graph also shows that the false alarm rate for our methods is between 4% and 6% for most of our methods (once again, using 0.19 as the threshold value generated an outlier - a very large false alarm rate of around 16%).

We also compare our methods with the following strategies and techniques:

- **IPAM**. This detector is based on single-step command transition probabilities, estimated from training data. The single-step method only looks back one command to determine the likelihood values. Our method may consider the

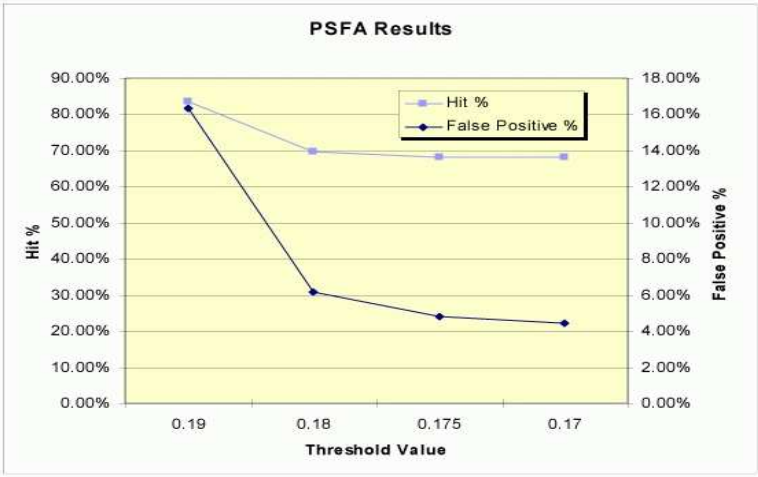


Figure 4.9: Performance with varying threshold values

dependence of several commands, looking back as long as the length of the trace. IPAM (Incremental Probabilistic Action Modeling) was developed by Davison and Hirsh [41] in an effort to predict sequences of user actions. In this method, the estimated probabilities are updated continually, using an exponential updating scheme. That is, upon arrival of a new command all transition probabilities from the previous command to another command are aged by multiplying them with an aging parameter, α.

Given a command, it is then possible to predict the next command by choosing the one corresponding to the highest transition probability. A prediction is labeled "good," if the next command turns out to be among the top four predicted commands. The fraction of good predictions of the test data forms the score and is compared against a predetermined threshold to cause alarms.

- **Bayes One-Step Markov**. This detector is also based on single-step transitions from one command to the next [48]. It uses a Bayes factor statistic to test the null hypothesis that the observed one-step command transition probabilities are consistent with the historical transition matrix (the mapping of one command followed by another). The authors form two hypotheses:

  1. The null hypothesis assumes that the observed transition probabilities

stem from the historical transition matrix.

2. The second hypothesis is that they were generated from a Dirichlet distribution.

They then test hypothesis (1) versus hypothesis (2) by forming the Bayes factor, the ratio of the probabilities of the data under the two hypotheses. Individual thresholds for the Bayes factor are developed for each user based on the average such score for the particular user and the average score across all users. A comparison of the Bayes factor with this threshold determines the decision of the detector.

- **Hybrid Multi-Step Markov**. This method, by Ju and Vardi [93], is based on Markov chains. Their model actually toggles between a Markov model and a simple independence model, depending on the proportion of commands in the test data that were not observed in the training data.

  – *The Multistep Markov model*

  The authors define k be the smallest number such that the most frequently used $k-1$ commands of user u account for at least 99% of that user's training data. All other commands, including those not appearing in user u's training set form a category labeled $other_u$. They then combine the most frequently used $k-1$ commands and $other_u$ to constitute the Markov chain's state space M. A log-likelihood metric is formed from aggregating probabilities from all elements in M and multiplying the aggregate by a specialized log function.

  – *The independence model* When the test data contain many commands unobserved in the training data, a Markov model is not usable. In such instances, the authors use a simple independence model with probabilities estimated from a contingency table of users versus commands. This model assumes that user u's commands are independently generated from a multinomial random distribution. To this end, they create a popularity score form user u's dataset.

To combine these methods the authors create two hypothesis:

- H:0 commands are generated by user u
- H:1 commands are generated by one of the other users

They then define log-likelihood-ratio statistics for both the Markov and independence models, which are combined in a piecewise manner to form the score to compare against computed thresholds.

- **Sequence-Match**. This approach is based on the early work of Lane and Brodley, refined in [104]. For each new command, the authors compute a similarity measure between the most recent 10 commands and a user's profile. A user's profile consists of command sequences of length 10 that the user has used in the past. The similarity measure is a count of the number of matches in a command-by-command comparison of two command sequences, with a greater weight assigned to adjacent matches. This similarity measure is computed for the test data sequence paired with each command sequence in the profile. The maximum of all similarity values computed forms the score for the test command sequence. Because these scores are very noisy, the most recent 100 scores are averaged. If the average score is below a threshold, an alarm is raised.

- **Compression**. The idea behind the compression approach is that new data from a given user compresses at about the same ratio as old data from that same user, and that data from a masquerading user will compress at a different ratio and thereby be distinguished from the legitimate user [142].

  The compression algorithm builds compression rules from the beginning of the file. Therefore, the test data is appended to the end of the training data in the same file. It is important that the the training data comes before the test data rather than the other way around. They use a score x to be the number of additional bytes needed to compress test data when appended to training data. There are several compression methods, many of which are based on the

Lempel–Ziv algorithm. The authors used the UNIX tool "compress" which implements a modified Lempel–Ziv algorithm popularized in [181].

- **Uniqueness**. This approach, due to Schonlau and Theus [141], is based on the following two ideas about command frequency:
  - Commands not seen in the training data may indicate a masquerade attempt.
  - The more infrequently a command is used by the user community as a whole, the more indicative that command is of being a masquerade.

  In this process the order in which the command appears does not matter. The authors define a uniqueness index which is 0 if all users have used this command before and 1 if none of the users has used it before (with corresponding values between). The uniqueness index is then multiplied by a weight representing the command usage relative to other users. It reduces the score contribution of commands that other users often use and this user rarely uses. Resulting values are averaged across the user's session resulting in a value to be compared with pre-computed thresholds.

- **Naive Bayes**. This approach, proposed by Maxion and Townsend [114] uses simple, probabilistic Naive Bayes classifiers known for their inherent robustness to noise and their fast learning time. This model assumes that the user generates a sequence of commands, one command at a time, each with a fixed probability that is independent of the commands preceding it (this independence assumption is the "naive" part of Naive Bayes). The probability for each command is based on the frequency with which that command was seen in the training set. The probability that a test sequence of the five commands [a,a,b,b,b] was generated by a particular user, u1, is given by Equation 4.5.

$$P_{u1,a} * P_{u1,a} * P_{u1,b} * P_{u1,b} * P_{u1,b} \tag{4.5}$$

  In Equation 4.5, $P_{u1,x}$ is the probability that user 1, (u1), typed command x. For each user u, a model of "Not u" can also be built using training data from

all other users. The probability of the test sequence having been generated by "Not u" can then be assessed in the same way as the probability of its having been generated by user u. The larger the ratio of the probability of originating with u to the probability of originating with "Not u", the greater the evidence in favor of assigning the test sequence to u.

- **Naive Bayes with updating**. This method, also proposed by Maxion and Townsend [114], is the same as the Naive Bayes method, but is enhanced with an updating scheme whose purpose was to accommodate drift in the data due to changes over time.

- **PSFA (x)**. This is the method previously outlined in this section where x is the average-based threshold value for the automata traversal. The higher the threshold value, the more liberal the detection will be. Liberal detectors usually have a higher hit percentages, and higher false alarm rates.

Figure 4.10 shows the functional relationship between the hit rate and the false alarm rate in a graph called a ROC (Relative Operating Characteristic) curve. In this curve, the higher and more leftward a point is, the better the method is considered to be. Determining which metric is more important, the vertical height or the horizontal length, depends on the preference of the application. For example, if an application needs to have a very small number of false alarms, it may choose not to employ the **PSFA(0.19)** method even though the method has the highest hit rate. Following [114], we use the cost formula shown in Equation 4.6 to compare our methods to the aforementioned list of methods.

$$Cost = \frac{\alpha M}{A} + \frac{\beta F}{N - A} \tag{4.6}$$

α	miss multiplier ($\alpha = 1$ in Maxion evaluation)
M	number of attacks that were missed
A	the total number of attacks
β	false alarm multiplier ($\beta = 6$ in Maxion evaluation)
F	the number of times the alarm was raised on the normal user session
N	the total number of sessions (N=10,000 for SEA data)

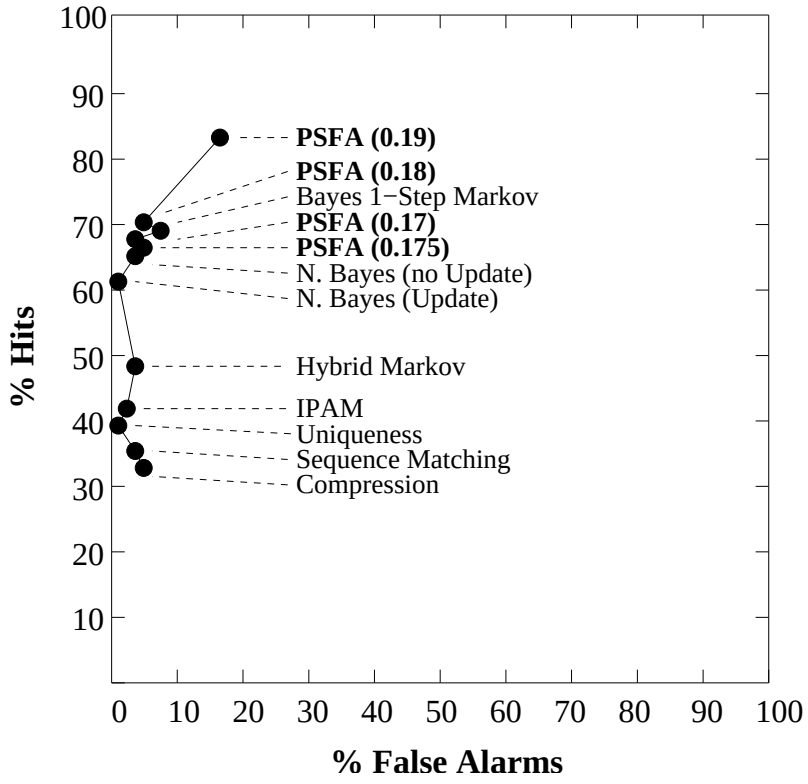


Figure 4.10: ROC

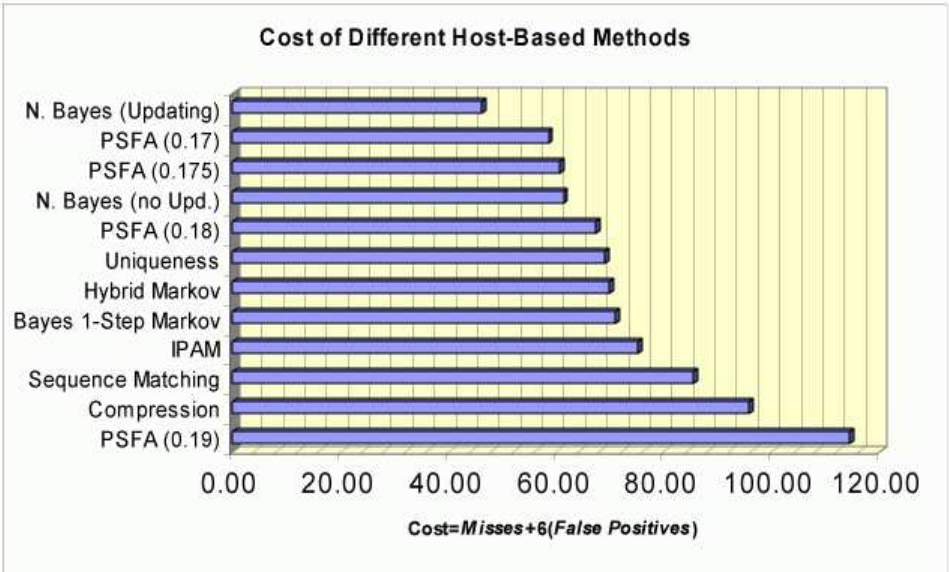


Figure 4.11: Cost of different host-based methods

Figure 4.11 shows the cost of our detection methods as well as other masquerade detection methods outlined in [114] and [142]. We should point out, that setting $\beta = 6$ places a large bias against false alarms. In more realistic environment, the penalty for false alarms may not be quite so high. Still, even with $\beta = 6$, although we are not the top method, we occupy 2 of the top 3 places and 3 of the top 5 places as shown in Figure 4.11.

Another comparison could be based on the following observation. There is a certain cost of missing an attack as well as another cost of a false alarm that interrupts user's and system manager's work unnecessarily. In our opinion, a ratio, let us call it ρ, of the first cost to the second one, can range from about 5 to 100. Hence, we propose the following in Equation 4.7.

$$C(M, F, \rho) = \rho\frac{M}{N} + \frac{F}{N} \tag{4.7}$$

The advantage of the new formula is that it does not depend on the frequency of attacks, as the Maxion formula does, but only on the relative cost of missing an attack versus a false alarm. If we let $f = \frac{A}{N}$ be the attack frequency, then Equation 4.8 expresses Maxion's cost formula in terms of the new cost formula.

$$\begin{aligned} \frac{\alpha M}{A} + \frac{\beta F}{N - A} &= \frac{\alpha M}{fN} + \frac{\beta F}{N(1-f)} \\ \frac{\beta}{1-f}\left(\left[\frac{\alpha(1-f)}{f\beta}\right]\frac{M}{N} + \frac{F}{N}\right) &= C\left(M, F, \frac{\alpha(1-f)}{f\beta}\right)\frac{\beta}{1-f} \end{aligned} \tag{4.8}$$

Note in Equation 4.8, ρ is computed to be $\frac{\alpha(1-f)}{f\beta}$. The α and β components of ρ are both constant coefficients, leaving the dominating factors of ρ to be $\frac{(1-f)}{f}$. Noticing that $\frac{(1-f)}{f} \approx \frac{1}{f}$ for $f << 1$ and $1-f$ for f close to 1, we see that the Maxion formula changes the relative cost of missing an attack depending on the frequency of attacks, from about $\frac{\alpha}{f\beta}$ for infrequent attacks to about $\frac{\alpha(1-f)}{\beta}$ for frequent ones. Less frequent attacks are more costly, compared to the cost of raising a false alarm, then more frequent ones. In contrast, our formula keeps the ratio of costs of missing attacks to those of false alarms independent of frequency of attacks.

To evaluate the other methods according to the new cost formula, we needed

to determine the number of attack blocks used in the SEA dataset. To develop an expected value for this statistic, a detailed analysis of how the dataset was formed must be conducted. Recall the details of the SEA dataset's block organization described in Section 4.2.7.1. Every user had 100 blocks of possibly contaminated data for testing purposes. The first contaminated block can appear at any of $s = 1, ...b = 100$ positions with the same probability pa, where $pa = 0.01$ is the given probability of new contamination. A contaminated block extents to the subsequent block with the given probability $pc = 0.8$ and is truncated at the end of the user input, i.e., at the block $b = 100$. Determining the average number of contaminated blocks leads to a complex recursive formula due to the dependent nature of the rules of the SEA dataset (shown in Equation 4.9, many of the steps to form this equation have been omitted for simplicity).

$$\begin{aligned} c(b) &= (1-pa) * c(b-1) + pa * pc^{b-1}\left[2b - 2 - \frac{1}{(1-pc)}\right] + \\ &\quad \frac{pa}{(1-pc)} + pa * (1-pc)\sum_{i=1}^{b-1} pc^{i-1} * c(b-i-1) \end{aligned} \tag{4.9}$$

b	number of blocks per user (100)
$c(x)$	average number of attacked blocks for any value of x under this assumptions
pa	probability of a new attack (1%)
pc	probability of a continuing attack (80%)

Because of the dependencies of this distribution, we wrote a program to simulate the number of attacked blocks given the distribution criteria. We ran this program many times, averaging the number of attacked blocks between runs to get the following estimate:

$$c(100) = 4.9256$$

To use Maxion's evaluation coefficients in our new cost formula, we would compute

ρ according to Equation 4.10.

$$\rho = \frac{\alpha(1-f)}{\beta f} = \frac{1(1-f)}{6f} = \frac{1(1-0.049256)}{6(0.049256)} = 3.217 \tag{4.10}$$

Figure 4.12 shows the new cost comparison for the top 8 methods using varying values of ρ. Note that all methods may be close at low values of ρ, but as the importance of insuring that a minimal number of missed attacks grows, the cost of different methods becomes more pronounced.

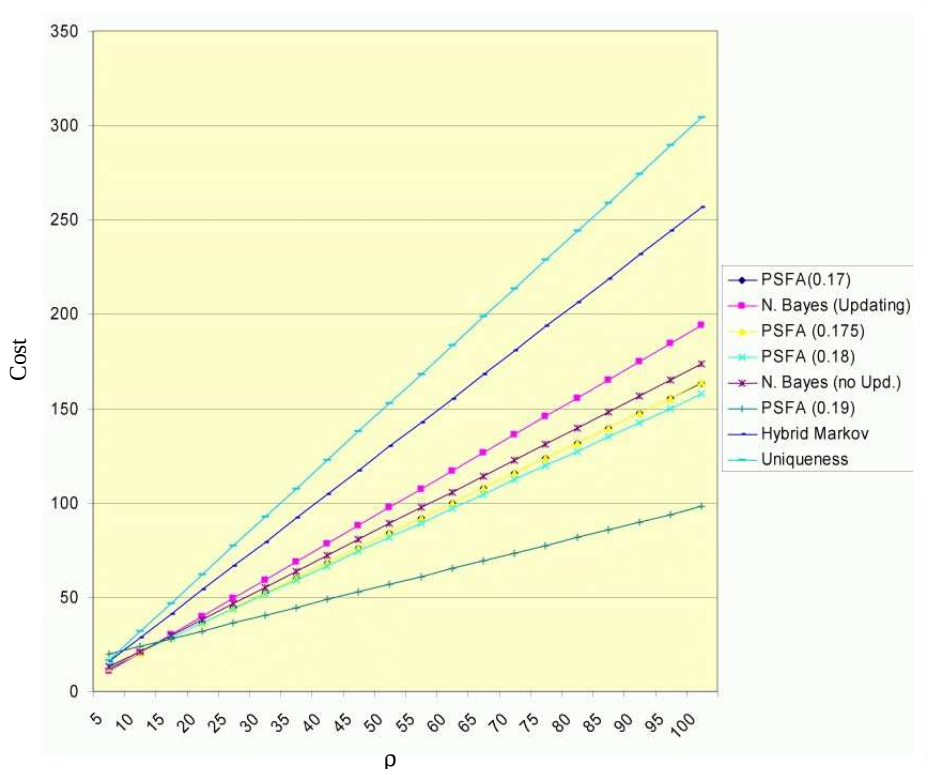


Figure 4.12: Cost of different host-based methods

4.2.8 Summary

In this chapter we have presented a host-based masquerade detection tool which is very successful at identifying anomalous user behavior. The strength of the tool is related to the signatures that it creates to store information about the frequent commands as well as their order. These signatures provide accurate models

of typical user behavior. Any user behavior that clearly detours from the signature will return low probabilities and raise an alarm. We have evaluated this technique against other methods using the SEA dataset and a fixed command window size. A method such as this could also be extended to monitor other types of behavior patterns including network or application specific behavior.

4.3 Related Works

Host-based intrusion detection has emerged as a very important part of the information assurance and security areas. Many research groups have made strides in this area and are discussed in this section.

4.3.1 General Host-Based Intrusion Detection

STAT (State Transition Analysis Tool) is a detection system which uses attack signatures to recognize intrusions [85]. In this system, intrusions are represented as a series of state changes from some secure state to a compromised state. The system uses audit data as evidence of state transitions, causing an alert if a compromised state is reached. They create their state diagrams based on the attacks they have seen, not the normal behavior of the user or root; therefore it is a misuse detector. The authors actually point out that their system is very inefficient in detecting masquerades, which is the focus of our intrusion detection system.

Model-based intrusion detection is another technique in which intrusions are defined as high level models characterizing the steps of several successful attacks [64]. These models are then converted to sequences of audit records corresponding to a particular attack. The attack sequences are used to anticipate what a user would do next if they were executing one of the modeled attacks. Therefore, the system would only have to search for those audit records which would come next in one of the currently active attack sequences. Because the attacks are modeled in this fashion, this is also a misuse detector.

A group at Reliable Software Technologies Corporation designed tools to profile the behavior of programs that use techniques ranging from memorization to generalization [65]. The first strategy used is a simple equality matching algorithm

for determining anomalous behavior. In the equality matching algorithm, sequences of BSM events are captured during online usage and compared against those stored in the database built from the normal program behavior profile, making this an anomaly detector. If the sequence of BSM events captured during online usage is not found in the database, then an anomaly counter is incremented. Thresholds are then applied to these anomaly counters to decide when to cause an alarm.

Authors of [65] also use a feed-forward back-propagation neural network for learning program behavior. Strings of BSM events are passed to a neural network in the order in which they occurred during program execution. The output of a neural network, that is, the classification of the input string, is then added into a leaky bucket. During each time-step, the level of the bucket is decreased by a fixed amount. If the level in the bucket rises above some threshold at any point during execution of the program, the program is tagged as anomalous. The leaky bucket algorithm is used to allow the neural network to pass judgment over the entire session when it only received a smaller sequence of BSM audit data. The authors also pursue the use of Elman neural networks which have a sense of state by using internal context nodes which in effect keep previous information in the neural network. This essentially solves the problem addressed by the leaky bucket algorithm used with the normal feed-forward neural network [65]. These neural network approaches are misuse detectors because they train on the intruder's behavior.

NNID (Neural Network Intrusion Detection) uses neural networks to predict the next command a user will enter based on previous commands [138]. Haystack, a combined anomaly detection/misuse detection IDS models individual users as well as groups of users [145]. It assigns initial profiles to new users based on the user's group membership, and updates the profiles once a pattern of actual behavior is recognized. In the Haystack system, user profiles are stored in a database simply composed of the audit trail of each user. The length of the audit trail is determined by a variable called the event horizon, which is defined as the number of audited events the audit trail analysis must "remember" in full detail at one time while processing a series of events recorded in the audit trail. EMERALD eXpert-BSM, a real-time forward-reasoning expert system, uses a knowledge base to detect multiple

forms of system misuse [128]. The forward-reasoning architecture helps eXpert-BSM detect intrusive behavior across multiple system event orderings while also accounting for specific pre- and post-conditions of those sequences.

Authors of [179] use a real-time "Shell Tracker" to monitor Basic Security Module (BSM) audit data and find users who illegally become root. This shell tracker uses a technique called bottleneck verification, in which all valid states of the user are modeled and form groups which are united by a few well-known transitions. Typically, one group is normal user activity and the other group is a super-user shell. Simply put, the bottleneck verification algorithm says that if the state of the user moves from one group to another without executing one of the well known transitions, an User-to-Root attack has occurred.

4.3.2 Masquerade Intrusion Detection

One of the seminal research papers modeling behavior for anomaly-based masquerade detection was presented by Forrest et al [60]. In this work the authors model the "normal" behavior of *sendmail* and *lpr* at the system call level. They use a sliding window approach to record what commands follow each other. Once a database of these relations is formed, new traces are checked against it using a similar sliding window approach. Results are expressed as the number of mismatches as a percentage of the total possible number of mismatches.

An extension of Forrest's work can be found in [177]. In this work the authors modeled the *named*, *xlock*, *login*, and *ps* commands, as well as the *lpr* and *sendmail* commands. They also explored the use of sequence enumeration, relative frequency methods, and Hidden Markov models for detecting anomalies in new traces.

ImSafe, a tool that has its roots in anomaly detection, monitors the system call traces produced by specific applications and tries to predict the next system call as accurately as possible [53]. First, ImSafe must go through a learning phase to construct a profile of the application to be monitored. Then, that profile is used during the detection process. Our approach is similar to that of ImSafe except that user behavior is modeled instead of application behavior.

The authors of [107, 106] use the k-Nearest Neighbor (kNN) classifier to or-

ganize program behavior into groups of normal and intrusive behaviors. In this case, each system call is treated as a word and the collection of system calls from each program execution as a document. The documents are then classified using the popular text categorization method, the kNN classifier. Once the training set is classified, a similarity metric is calculated between the new process and those in the training set. The average similarity metric of the k nearest neighbors is compared to a predetermined threshold. If the average is greater than the threshold, the behavior is labeled normal.

Recent works focusing on modeling user behavior for detecting masquerades can be found in [142] and [114]. In these two works, the authors investigate and present results from several masquerade detection strategies. Short descriptions of these methods as well as comparisons between their results and results from our systems can be found in Section 4.2.7.3.

CHAPTER 5
Congestion Control

Network congestion can result from a number of different factors. In a network, if the packets flowing to an intermediate router exceed the router's capacity, the router's queue will begin to build up. If there is insufficient memory to hold all of the queued packets, some will be dropped. A simple graphical illustration of the packet delivery performance during congested time periods can be found in Figure 5.1.

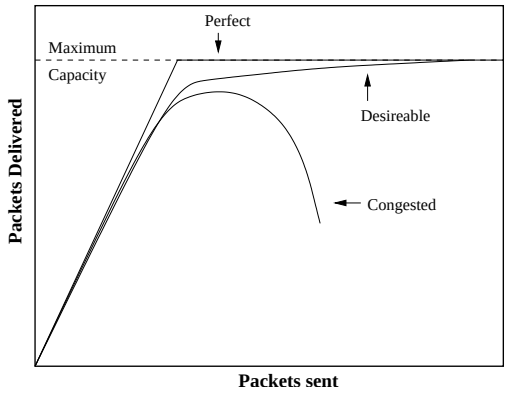


Figure 5.1: Graph of packet delivery during times of congestion [163]

Adding more memory may temporarily help, but it has been shown that if routers have a very large memory, congestion could get worse, not better. This is because many transmission protocols use a timeout value in which the sender will resend any packets which have not been confirmed as received. By the time queued packets reach the head of the queue, their timeout period would have already expired (possibly multiple times), and duplicates sent. These packets would still be forwarded to the next router, clogging up the network all the way to the destination [121].

Congestion is furthered assisted by combinations of slow processors (router processors) and low-bandwidth lines. Many times upgrading one without upgrading the other only moves the bottleneck from one network segment to another part of the network. Even when these components are upgraded uniformly in one network,

connections to or from other paths can cause queue buildups when components in neighboring networks do not have the same capacity. Another related cause of congestion is malfunction of the equipment. When mechanical problems are present, equipment will often lose functionality unpredictably. Congestion, once created in a part of the network, may also spread to other parts of the network. Naturally, different types of networks and networking protocols have different ways to deal with congestion.

In this chapter, we will first describe the current standards in the field of congestion control. We will then discuss our system for congestion arbitration and source prediction using neural networks. Lastly we will review related research efforts in this area.

5.1 Current Standards

The following congestion control methods will be discussed according to the network protocol layer to which they pertain in the OSI reference model [42].

5.1.1 Transport Layer

In networks today, congestion problems are most often handled with transport layer protocols. The most prominent among them is TCP, which uses acknowledgments for data received by the destination. If the sender does not receive an acknowledgment for the recently sent data, the sender assumes that the data was lost because it was sending at a rate which the network could not handle. Thus, it will drastically reduce its sending rate, a process referred to as exponential back-off. This mode of operation allows routers in the path of the transfer to intentionally not forward a packet to the destination in order to slow down TCP sources. The sender will slow down and re-send the unacknowledged packets of information. Some versions of TCP, such as TCP Vegas, also attempt to predict congestion before packet loss occurs by analyzing the round trip times of the packets [22, 2]. In these cases, the longer the round trip time, the greater the congestion in the network. More information about the specifics of TCP congestion control can be found in Appendix A.

5.1.2 Network Layer

Other efforts are made at the network layer to avoid congestion in a couple of ways. These methods require network-layer components (routers) to determine or predict congestion in the network, and then provide notification to the sender indicating the congestion.

5.1.2.1 Congestion Detection/Prediction

Congestion detection is simply the process of noticing the results of network congestion. This can be done by monitoring the number of dropped packets or the queue size of the networking components. However, simply detecting the after effects of congestion is not so helpful. Predicting congestion or predicting congestion effects could enable the network manager to prevent or lessen the effects of congestion.

Many of today's strategies for predicting and possibly stopping upcoming congestion rely on closely managing the queues maintained in the network layer components. This has sparked a discipline known as Active Queue Management (AQM). These strategies typically allow routers or other intermediate nodes to monitor their network traffic queues for an indication of network congestion. RED (Random Early Detection or Random Early Drop) is an example of one strategy which monitors queues in an effort to prevent the queues from overflowing causing massive packet loss. According to the RED algorithm, routers associate a "packet drop rate" with each queue. This rate is nominal when the queue length is below a certain threshold value. However, it increases gradually along with the queue length. If the queue length reaches its maximum size, the packet drop rate is set at 100%. The drop rate is actually the rate at which the router will randomly drop packets. This is useful, because transport layer protocols normally consider a dropped packet as a sign of congestion in the network and therefore reduce their sending rates. It is important to note that dropping packets in RED algorithm starts after the queue length has passed a certain threshold, not when the queue is full. This gives this strategy the flavor of "early detection" [21, 82]. FRED (Flow-based Random Early Detection) is similar, but determines detection on a per-flow basis. This allows the algorithm to analyze the network based on who may be causing the most trouble, or which flows

may be most important. FRED is considered by many to be a "better RED" [108]. Research into an alternate active queuing mechanism is also being conducted at the University of Michigan. A technique called BLUE is used to determine network congestion based on a probabilistic model. Packets will be dropped based on the following changing probability. When a router starts routing packets, the probability is low. However, as packets are dropped due to full queues or if a threshold is passed, the packet dropping probability is increased by a predetermined delta. However if the queue is empty for a period of time, it causes a link idle event which decreases the packet dropping probability by a different predetermined delta [56]. References for many other AQM schemes can be found in [58, 33].

5.1.2.2 Congestion Notification

This notification can be a direct notification in which the router sends a choke packet informing the sender of the congested network. An example of this notification is in the ICMP (Internet Control Message Protocol). If a node is recognized as sending too much data to a particular router, the router may disregard this node's packets and send an ICMP SOURCE QUENCH message to the sending node. When the sending node receives this message, it is expected to slow down. This technique is not very common because when congestion becomes a problem, a flood of these feedback packets make the situation worse. In addition, in transport layer protocols such as UDP (User Datagram Protocol), the feedback packets are frequently ignored anyway [163].

Network-layer components may also indirectly inform the sender of the congested network by marking or updating a field in a packet flowing from the source to destination. Upon receipt of the marked packet, the receiver would notify the sender of the congested network. This type of approach has been used in the IBM SNA architecture [143], the DEC DECnet architecture [89], ECN (Early Congestion Notification, described in 5.1.3) in TCP/IP networks [59], and in ATM ABR (Available Bit-Rate) congestion control [90].

5.1.3 An Example Involving Transport and Network Layers

One common combination of the strategies mentioned in this section is using AQM with an ECN-aware transport layer protocol. To keep Active Queue Management schemes and other strategies from having to physically drop packets to signal congestion, Early Congestion Notification was created to be inserted into the network and transport layer protocols. In this sense, Active Queue Management and ECN really go hand in hand. The active queue management system determines when congestion is present or will be present at routers and intermediate nodes. The transport protocol used between sender and receiver must then support ECN, so the receiver can inform the sender of the congestion detection. Dropping packets to signal congestion is not necessary because the complete buffer size of in-route networking components may not have been reached. Using ECN, techniques such as RED would just set a Congestion Experienced bit in the packet headers of packets it sees when it wants to let senders know about congestion. When a sender receives a packet acknowledgment with the Congestion Experienced bit set, it is required to function as if the packet was lost (without re-sending the packet). For TCP traffic, this would mean slowing down its sending rate. The formalization of these procedures are detailed in [133].

ECN and Active Queue Management are widely used as control measures for data communications networks. Active Queue Management and ECN support is available on several BSD derivatives (FreeBSD / NetBSD / OpenBSD) as well as on Linux and AIX. Support is also available in Nortel's Open IP Environment and in the widely used ns-2 network simulator [59].

5.2 Neural Network Congestion Arbitration and Source Prediction

Our congestion arbitration system uses neural networks to not only detect or predict network problems, but to also find the source of the problem. Once the source of the problem is determined, it can be fixed before it surfaces, or shortly thereafter. At present, our neural network is trained with samples from the network, using an expert (network administrator) to determine where problems will be caused

in our network due to existing conditions. During the training process our network administrator can initiate several training sets and easily see where the problems occur. The network is trained on a relatively low number of samples and proves general enough to give good results.

5.2.1 Architecture

A high level view of our architecture reveals a network with a control agent existing somewhere on a node in that network. This control agent has both, the power to collect performance data from network nodes, and the power to influence those network nodes. Data collection can be accomplished in two ways: either the nodes involved would report the necessary statistics to the control agent or the control agent would poll these nodes.

5.2.1.1 Data Network

Optimally, we would have tested the system on a local computer lab or a testing lab put together for this purpose. However, neither of these was available at the time of project development. In the absence of the needed testbed, *NS*, a discrete-event network simulator targeted at networking research, was used to model the network and different scenarios of network traffic (NS can by found at http://www.isi.edu/nsnam/ns/) [12]. NS simulates network architectures on a packet by packet basis, giving the user the ability to monitor very specific as well as aggregate statistics about all aspects of the network activities. Using a simulator, of course, makes the integration of a control agent easier, but a similar design could be implemented on a real network.

In our example, the network consisted of several nodes in a configuration where all of the network nodes were attempting to send data to one node (see Figure 5.2). Each node attempts to send at a random bit rate. A random amount of variance is given to each node's rate to better represent traffic in a real network and possible traffic coming in from other nodes outside of our simulation. Link capacities between sending nodes were given arbitrary values (described in a later section) for testing purposes. Some links were able to handle much more traffic than others.

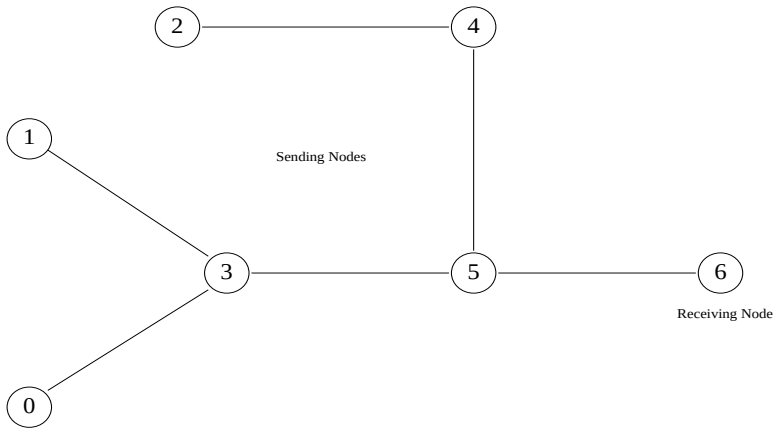


Figure 5.2: Network topology

5.2.1.2 Control Agent

We create a control agent containing a neural network that is trained prior to being placed in production. In our simulation, the control agent is called at a regular interval in part of the simulation code. This enables the agent to easily monitor and influence traffic statistics from each node. The control agent gathers information from each managed node, performs several mathematical functions normalizing the values, and makes a decision about where, if anywhere, network problems will occur. having the prediction, we can take steps to stop or prevent it.

If it were to be implemented on a real network, SNMP traffic variables on the IP and Interface levels could be used to gather the traffic data. Classifications and service level agreements, communication pipes, or ECN could be used to implement the control aspect. In this case, the control agent must reside on a node close to the network which has access to all nodes being monitored in the network. This keeps our control agent local and would prohibit implementation in a very large network without changing the architecture. Such an architecture extension will be described later.

5.2.2 Implementation Details

The system consists of three separate programs, one implemented in Tcl, the language used to run simulations in NS, one implemented in C for file manipulation, and a third actually running the neural network. We use the publicly available

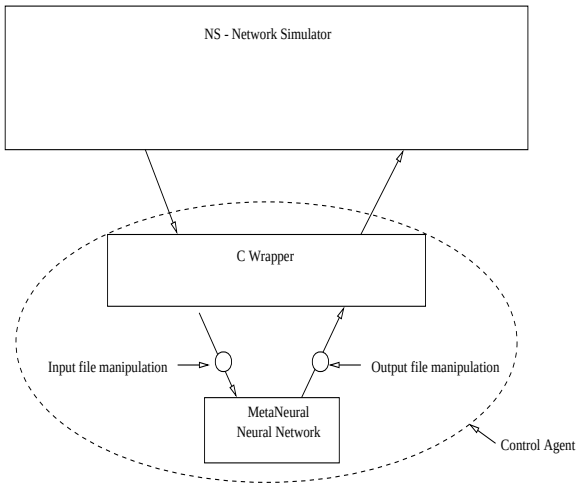


Figure 5.3: System component and flow diagram

MetaNeural Neural Network application as our neural network (MetaNeural can be obtained from http://www.drugmining.com/). MetaNeural is a general-purpose back-propagation neural network code. These programs communicate with each other via files to synchronize the running of NS with the running of the neural network program that determines if a current network configuration might cause a problem. An illustration of these relationships is shown in Figure 5.3.

5.2.2.1 The Simulated Network

The simulated network is arranged in such a way that six sending nodes are connected to one receiving node through several links which direct the packets to the destination (Figure 5.2). The sending nodes produce data in a way similar to User Datagram Protocol (UDP) agents, sending constant bit rate (CBR) traffic with a randomized parameter to add variance to the traffic. In NS, each connection is explicitly stated and each sending agent in each node is configured to send to a particular receiving agent. To determine how fast the sender sends data, the packet size and a packet interval are given in the simulation script that defines the simulation run. The sender sends a packet of the designated size at the designated interval. The receiver simply has a null agent that receives the data and sends no responses. NS Queue Monitors are attached to the queues to keep track of the status of each queue. We gather statistics such as packets received and the size of the queue during the

simulation. During the simulation, the control agent executes at a regular polling interval, monitoring the traffic and making decisions. Files are created for each node to keep track of that node's data. During each run of the network simulator, the files are extended with the new data from the latest interval. The most important part of the control agent is the neural network prediction module. For our control agent implementation, we used a single hidden layer, feed-forward neural network. This was a compiled application, so wrappers were needed to control the input and output dealing with the neural network. The wrapper program is written in C and is called, after the data files are updated, by the simulator to normalize and run the data files through the neural network.

C Wrapper

As mentioned before, to execute the control agent, a C wrapper is first called. This is where the bulk of the calculations for the neural network program are done. The C wrapper first opens the files written by the simulator, which contain historical and current values for the number of packets. The program uses these values to compute the average number of packets, the variance of packets, and the third moment given the appropriate polling interval. In the first iteration, the average is the current number of packets and the variance and third moment are zero. These values are then normalized for the neural network using the basic normalization function found in Equation 5.1.

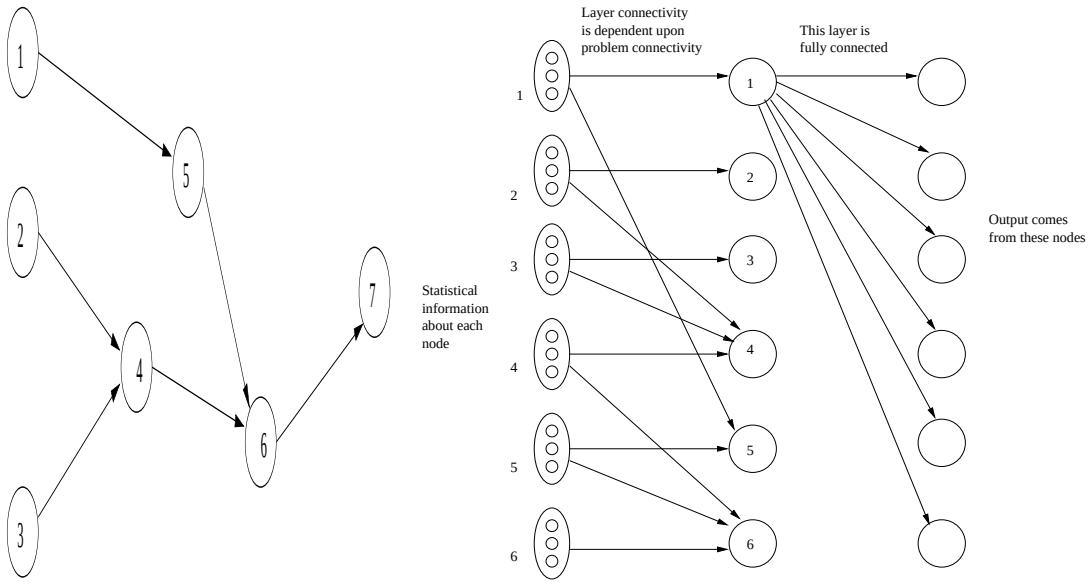
$$value_{normalized} = \frac{(value - value_{minimum})}{(value_{maximum} - value_{minimum})} \tag{5.1}$$

The normalized values are then combined into one input file to the neural network package for a decision. The neural network program is executed using new input files and the output is rendered in yet another file. This file is read by the C wrapper and converted into an NS readable format for the simulator to process.

Neural Network Specifics

The neural network used by the control agent has 3*n input nodes, 1 hidden layer containing n nodes, and n nodes in the output layer. The 3*n input nodes

correspond to the n traffic generating nodes in the network simulation; each network node is represented by three input layer nodes corresponding to the average number of packets, variance, and third moment of traffic at this network node. In our example (Figure 5.4(a)), n is six because only six nodes contribute traffic to our network.



(a) Sample data network topology

(b) Neural Network architecture modeling the architecture of the sample data network.

Figure 5.4: Relationship between data network topology and the neural network structure.

To enforce the importance of adjacency relationships between nodes in the data network, we place an additional optimization on the structure of the neural network. The weights are pruned to the point in which the neural network reflects the connectivity of the actual data network. The n nodes in the hidden layer also represent active nodes in the data network. Instead of providing a fully connected environment between the input layer and the first hidden layer, we only allow connectivity from input neurons that represent nodes adjacent to represented nodes in the hidden layer. An example is shown in Figure 5.4(a) and Figure 5.4(b). In the model of the neural network in Figure 5.4(b), the hidden layer is representa-

tive of the participating nodes in the data network. The statistical data regarding each node is provided to the hidden node representing the actual node as well as to the hidden nodes representing the actual node's neighbors. This is continued for all first layer nodes of the neural network. As a result, the statistical information from node 1 is given to both the hidden node representing node 1, and the hidden node representing node 5 (1's neighbor). This process is important in realizing the relationships between adjacent nodes in data communication networks.

The output of the neural network is a mask representation of which nodes are suspected of causing the problem. For example, an output of "010000" means the second node in the network is responsible for the network congestion. An output of "010100" indicates the second and forth nodes are both to blame, and "000000" means no problem threat is detected. The neural network is trained off-line, which involves creating a pattern file from which the neural network learns about congestion. Eighty-eight patterns are used to train the network; half are samples containing no network problems and half have congestion problems at various locations. In training the neural network, early stopping is used, allowing the training to go for about 15000 iterations. In this case, the least squares error is equal to or less than 0.04%.

5.2.2.2 Control from the Agent

As stated before, in NS, both an interval and a packet size are provided for agents sitting at the sending nodes to determine bandwidth used. The agent sends one packet at every interval, therefore the smaller the interval, the higher the bit rate. If our neural network predicts that a particular node is responsible for congestion, we conclude that the predicted problem source is using too many resources. To correct the predicted cause node's traffic rate, we add to its sending interval a small δt, thus reducing its bit rate. For example if node 1 is predicted as the problem source, Equation 5.2 would explain how node 1's bit rate would be corrected:

$$Interval_1 = Interval_1 + \delta t \tag{5.2}$$

The value of δt was chosen to be small compared to the flow time scale, because

we do not want to take the chance of over-correcting or even worse, if our prediction is wrong, to apply a large correction to the wrong node. To compensate for the small δt, the interval in which our control agent executes is also relatively small. Therefore many of these small corrections can be applied, correcting the problem slowly without drastically changing any one node's level of service.

We can also vary δt depending on the source of the traffic. This would allow us to institute certain Quality of Service rules into the system by assigning priorities to more important traffic. Traffic from more important sources would be reduced by a much smaller δt than the best effort traffic.

5.2.3 Test Cases

5.2.3.1 Testing Environment

The tests were performed on Ultra-SPARC 10s running Solaris 5.6. The packet size was set at 500 bytes for each sending node, most of the links between nodes were set to 5 MB/s and the link between nodes 5 and 6 was set at 10 MB/s. The delay for the links was set at 10ms and all queues implemented Stochastic Fair queuing (Equally Fair queuing). The interval was varied between 0.00100 and 0.00200 seconds; if the interval is 0.00100, a packet is sent every 0.001 seconds. The traffic generated is constant bit rate traffic, but the random parameter was set for each sending node so that the traffic would not be constant.

5.2.3.2 Results

A general breakdown of the results can be found in the graph of Figure 5.5 which shows that our current application detects and corrects congestion in about 90% of the cases. Our tests include cases in which corrections to one node are required, corrections to multiple nodes are required, and some where no correction is required. Failing includes either missing congestion or predicting congestion when there is none.

We ran thirty-one network simulations. Roughly 33% of the cases were simulations of a network without congestion problems. In these cases we want our control agent to realize that it does not have to do anything. The detector realized that there was no correction needed in all but 1 case. In this isolated case our agent

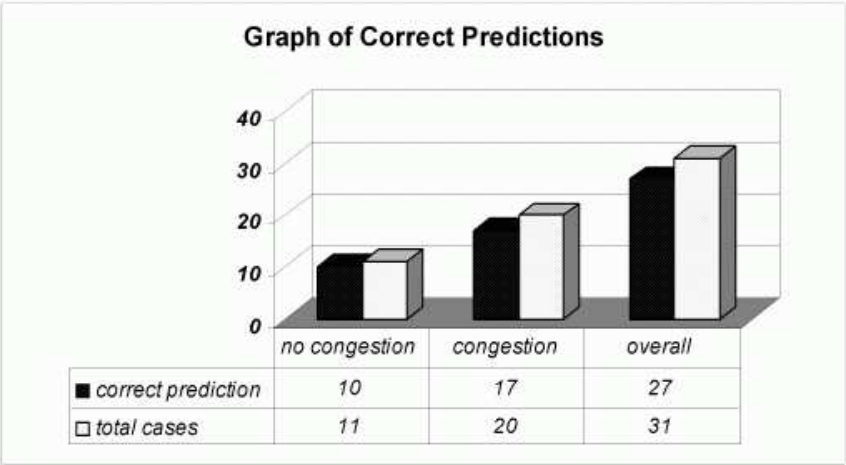


Figure 5.5: Graph of prediction results

unnecessarily applied a single small correction to a single node. The correction that the control agent applied was with a single δt, and therefore was minimal. About 66% of the total cases had various levels of congestion in various locations in the network. Of the congested cases, we were able to predict the cause and fix the problems 85% of the time.

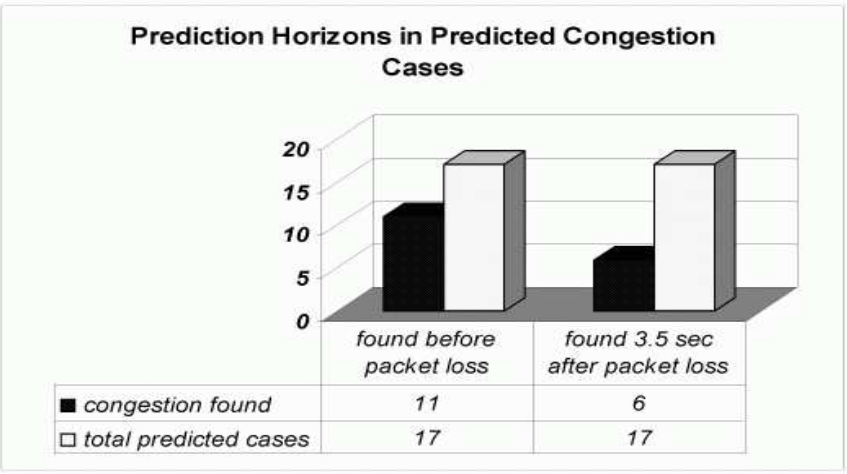


Figure 5.6: Graph of prediction horizons

The time frames in which we were able to detect the congestion problems can be found in Figure 5.6. Figure 5.6 shows that 60% of the time when we detected congestion, we were able to fix the problem before packets were dropped in the network.

We obtained this metric by simultaneously running the simulation with our detector and the simulation without the detector, and observing the differences in behavior. These were truly remarkable results, because the congestion was completely eliminated before it occurred. In the cases that we could not stop packets from dropping, we were able to return the network to a stable state within 3.5 seconds after packets began to drop.

Finally, in the cases that our detector missed the threat of congestion, there was a common characteristic. The neural network had trouble detecting congestion when a single node in a particular part of the data network caused a problem. This probably can be improved upon close examination of the training patterns and structure of the neural network.

5.2.4 Summary

In this work, we have illustrated, through the use of a network simulator, that a neural network can achieve great accuracy in predicting congestion. We realize that many more problems exist for which this approach is applicable, but predicting congestion is just one of the steps towards our research goals.

This particular network manager illustrates two of the strategies we want to show as necessary for efficient network management, namely:

- *Attention to appropriate time scale:*

  Network congestion is a problem that changes very quickly. Any algorithm addressing congestion would have to converge on a solution in an order of milliseconds or a few seconds. If care is not taken, the decision may be rendered after the problem has changed to a degree that the decision is no longer relevant to the environment. Once trained, a neural network can render decisions very quickly (i.e. in the time it takes the machine to perform the short series of floating point operations).

- *Incorporating all available data into management structure:*

  We also have shown a case in which a carefully constructed neural network can achieve above average results when structural information about the ac-

tual data network is used to form the connections between layers of the neural network. This special design forces the neural network to consider the relationships only of those nodes that we think are important.

A learning mechanism can be of great value for a network manager. The generalization power of a neural network particularly is appropriate because of the unpredicted variance of parameters that the network manager encounters. This work has shown how neural networks are an appropriate mechanism for decision making in pro-active network management.

5.3 Related Works

5.3.1 Applications of Learning Methods for Network Management

Approaching network management issues with learning methods is not uncommon. The following discussions show the wide variety of networking problems being approached with learning methods. While all of the references do not address the networking problems we focus on, they do illustrate the viability of learning methods to networking in general.

Topology and Routing

A group at Prairie View A&M University developed a Neural Network-based system, which looked at a network's traffic patterns and proposed a new physical configuration of the network. An administrator could then implement this new configuration by physically altering the interconnection of components in the network. Their goal was to reduce the amount of data flowing around the network, thereby decreasing the response time, and reducing the error rate [120].

Another group at the University of Maryland [66] is using neural networks to route traffic in Multistage Interconnection Networks (MINs). MINs are often used in parallel processing and distributed computing systems. In this work, the authors used a model of a distributed computing system as the model for the MIN. They use a Neural Network that functions as a very robust parallel computer generating routes faster than conventional routing approaches. The robust nature of neural networks was one of the factors that attracted the authors to this approach. The

authors propose creating a hardware version of a neural network router, utilizing Hopfield neural networks which, through carefully organizing optical emitters and detectors on opposite sides of a lens mask, can be implemented using an optical subsystem of an optical router [40, 54]. The neural network will be designed to model the topology of a MIN in which all layers are not fully connected. For example, if the MIN has three layers, it will have a three layer route array (to get from node 10 on the first layer to node 4 on the third layer, the route array may look like {10 13 4}, meaning that the path goes through node 13). Each $a_{i,j,k}$ of a routing array is represented by the output voltage of a neuron, $V_{i,j,k}$. If this value is 1, the path (i, j, k) will be used to route traffic from i to k. Such a close integration of a neural network with the optical hardware in the router will certainly promote and simplify use of neural networks in routing and congestion control, but hardware implementation issues of this kind are beyond the scope of this thesis.

Bandwidth Allocation

Neural networks have also been used as a very fast optimization approach to allocating bandwidth in a telecommunications network [24]. Inputs to the neural network in examples discussed in [24] were 13 link capacities and 42 traffic demands (total of 55 inputs). The neural network then returned 126 outputs representing capacities assigned to three paths for the 42 node pairs. To determine a training set for the neural network, the authors used a linear programming optimizer (a very slow process, but used only for training) to label the datasets. This solution does not deal with many of the dynamic properties of a communications network. If their model is slightly wrong or dynamics change for any reason, the system could produce less than optimal results with no feedback to adjust to these changes. This addresses a different problem than our research of predicting network congestion, but it does focus on delivering very fast optimization to the network, a quality we too strive to obtain to facilitate real-time optimization.

Connection Admission Control

Similar to the above-mentioned research, many authors use neural networks

for fast results, for once the neural network is trained, decisions and results can be obtained almost immediately. One of the drawbacks of this technique is the training process. Many times the neural network can converge very slowly. In [166], the authors advocate using a modular neural network instead of the traditional multi-layer Perceptron neural networks. Their research involves predicting a cell loss ratio in ATM (Asynchronous Transfer Mode) networks for the use of Connection Admission Control (CAC) schemes. These schemes are used to provide Quality of Service levels in ATM networks. Although this approach is similar to ours, it strictly deals with CAC schemes for ATM networks. The authors focus on predicting ratios of cell loss, while we focus on predicting cause and locations of network congestion. In a very similar work, neural networks are used in [126] to decide if accepting a new connection would violate quality of service constraints. This is another version of Connection Admission Control. CAC uses various traffic characteristics to make decisions; therefore, a version of these tools could be embedded in our agent which also needs traffic characteristics, or vice versa [170]. CAC is used to ensure agreements made between customers and service providers are fulfilled. Quality of Service measures will also need to be addressed in our tools to facilitate growing needs for this feature in applications.

5.3.2 Congestion Prediction and Avoidance Methods

In [74], a procedure for early detection of network congestion is described based on observed values of the mean aggregate loads (λ) and the mean capacities (μ). More specifically, the authors define a "congestion epoch" as any period when the instantaneous queue length exceeds a queue length bound which should be defined larger than the maximum steady state of queue fluctuations. In other words, a congestion epoch can only happen when $\lambda > \mu$. Because this determination occurs without (or before) packet loss, it is a form of early detection or prediction. Determining the maximum steady state of queue fluctuations, however, can be a difficult task.

The goal of the work done in [74] is to push congestion out of local networks to the edges where it can be dealt with by several ingress nodes rather than the

one overworked internal or egress node. The authors describe two schemes for congestion detection. The first deals with detection with the help of statistics from the internal nodes. The internal nodes simply set a bit in packets whenever the instantaneous queue length exceeds the abovementioned threshold. However, in the second scheme, the interior routers are not directly involved. Instead, the edges rely on the observation that each flow's contribution to the queue length, q_i, is equal to the integral $\int(\lambda_i(T) - \nu_i(T))dT$ (where ν is the output rate of flow). Values for ν can be computed at the receiver (egress node) and sent back to the sender (ingress node), or computed by the sender through carefully planned control messages sent to the receiver. Values for λ can be easily computed at the ingress node. If this accumulation is larger than a predefined threshold, the flow is assumed to be at the beginning of a congestion epoch. The end of the congestion epoch is detected when a one-way delay sample comes close to the minimum one-way delay. This method will work nicely especially for large, highly segmented networks. To contrast this method with the solution that has been proposed in this document, it should be pointed out that this method relies on aggregated flow information to and from a region of a network. Our work uses the detailed flow information from several nodes and the neural network learns to recognize the patterns leading to congestion among them. Hence, it could be said that the method in [74] uses the aggregated flow information for a network region whereas ours uses the detailed flow patterns for a group of directly connected routers. Both methods can be used to discover when congestion is imminent and take measures against it. The measures taken against detected congestion in [74] is dropping packets of the excessive flows at the edge of the region. An interesting study could be formed by comparing the efficiency and precision of congestion prediction provided by these two approaches.

Another group at Rensselaer Polytechnic Institute, is doing research on detecting changes in traffic patterns using a sequential Generalized Likelihood Ratio (GLR) test. They first gather time series MIB variable data using SNMP. This data is split into time windows of 2.5 minutes each. Using these windows, a sequential hypothesis test was performed using the Generalized Likelihood Ratio to determine the extent of statistical deviation between two adjacent time windows.

Once changes are detected using the GLR, the authors explored two ways of correlating the different alarms to values of the several MIB variables. They first try a Bayesian belief network whose model is based on a directed graph that spatially represents the hierarchical structure of the MIB variables. Their second technique was a duration filter which would correlate the propagation of many alarms with the variables' dependencies during a certain duration period [168, 167]. For example, one of the variables may be the amount of data received on a particular interface while another variable could be the amount of data received at the IP level. While these variables are both important, the amount received at the IP level is a direct result of all of the data received from every interface. Dependencies such as these are defined in [115]. Because of this dependency, alarms at the interface level could many times aggregate to an alarm at the IP level. Using these techniques, the authors were particularly successful in detecting when the network file system (NFS) stopped responding, a network fault confirmed by the system logs. The prediction horizon ranged anywhere from an hour before NFS crashed to 15 minutes after the crash. The approach used by these researchers is similar to the approach of our detection/prediction process. However, they used statistical methods for detecting patterns leading to faults in the NFS, while we proposed different learning strategies to learn the patterns leading to network congestion. Their technique focuses on determining faults using statistical data from one source on the network while we consider all the points of the network when determining congestion. Again a comparison between the method presented in [168] and ours is needed and again the difference is in the input to the congestion detection algorithm and in the approach to congestion detection. Our method collects the traffic patterns (packet delays and the higher moments of the delay distribution) for a collection of directly interconnected routers, whereas the authors of [168] rely on the selected interface statistics of a single router. They also use signal processing based algorithms to process their patterns, whereas we are using a neural network for our more complex input. Hence, it is fair to say that our approach is in between the previous two techniques presented in [168] and [74], wider in scope than the techniques of [168], but more localized than the techniques of [74].

5.3.3 Control Methods

In our system, we control the rate at which the sender transmits by using a hard limit set at the sender. In our simulated environment, this is a simple task; however, in a real network it is a bit more complicated. There are methods available for us to set transmission limits at the host (e.g. ipshapers, communication pipes); however, because they may enforce control at different layers, their performance may not always be as expected. There has been a great deal of research in the area of control methods. Many methods simply manipulate existing protocols by adding a couple of changes. Some of these methods operate on a per flow basis while others operate on an end to end basis.

Altering Window Sizes

In [11], a technique for transparently augmenting end-to-end TCP performance by controlling the sending rate of a host is discussed. In this work, TCP headers are altered in each acknowledgment packet to curb the sending rate of the sender to an arbitrary rate using a calculation of the round trip time. In this sense, the technique is a form of control, a way to limit the sender to the rate we want it to send. The calculation of the optimal rate is not the focus here.

In a normal TCP connection, the receiver passes its receiving window size (*recwnd*) in each acknowledgment. This is the way the receiver can keep the sender from exhausting its receiving buffer and give the receiver time to process the data received. This value, *recwnd*, is usually directly proportional to the amount of space remaining in the receiving buffer. If the buffer is full, this value could be zero. The sender will adjust its sending window (*cwnd*) as to never exceed the advertised (*recwnd*). The authors of [11] use the *recwnd* to enforce a given rate of control to a TCP source. In this method, the *recwnd* field of each acknowledgment packet is altered as it passes through the network to be the minimum of the existing window size (*recwnd*) and the window size determined by the following formula:

$$W_{new} = rate * t \tag{5.3}$$

In the above formula, rate is the desired rate of the sender (to be determined by a

separate control application), and t is set to be the round trip time of the connection [11]. The choice of the round trip time is optimal because a TCP source must wait for an ACK before advancing its *cwnd*, a process which takes approximately the same amount of time as the round trip time. The round trip time can be measured by observing timings of packets and corresponding ACKs without introducing new traffic to the network. This above calculation sets the window size to a quantity that will force senders to adhere to the rate asked, unless, of course, the connection is already operating at a slower rate.

Since packets must be intercepted between receiver and sender, this small change in each acknowledgment (ACK) packet requires a device (a computer in the middle) to be able to open and change each ACK packet. Once the window size information is altered, the checksum for the packet also must be altered, or the sender will assume a transmission error occurred with the ACK packet.

In addition to altering *recwnd* values, TCP rate control techniques also temporally pace the acknowledgments over a round trip time period. This process further smoothes the traffic and institutes control because a TCP sender must wait for an acknowledgment before continuing. If this method is also implemented, the machine in the middle will simply hold the ACK packet for a small period before allowing the modified ACK to continue. This ACK packet alteration must be done only once on the way to the sender. This method requires no changes to the sources or receivers in the network, but does require a machine to be dropped in the middle of the network with the ability to read and possibly alter all data moving across the network. This method also assumes that all TCP sources adhere to the window size field in the ACK packets. This may not be the case for some misbehaving-TCP sources created by hackers to cause problems [139]. The authors of [11] try to reduce the bursty characteristics of the traffic through this method of adjusting window sizes to control flows.

Adjusting the window size in acknowledgment packets is a well known way to alter sending rates and improve the TCP protocol. In [5], the authors show that in many cases changing the initial window size that TCP uses improves performance of TCP traffic. This is most evident for TCP flows involving small transfers, for

which the slow start of TCP can really hurt performance and overall capacity usage. The authors set the initial window, *cwnd*, at the beginning of the TCP connection to $4 * (maximum\_segment\_size)$ compared to $1 * (maximum\_segment\_size)$ used in the normal slow start algorithm. The current window continues to be reduced to 1 segment is a particular segment is not acknowledged before a timeout occurs.

Other Control

Many efforts in controlling sending rates from within the protocol involve altering windows sizes in the packet headers. Others, whose focus usually is not just about control, sometimes use methods outside of the protocol to adjust the sending rates of sources. A good example of this approach can be found in [74], in which a leaky bucket rate shaper is used to dynamically impose rate limits on the sender. The leaky bucket method of flow control involves maintaining a counter for a given traffic stream entering the network. Each time a packet enters, the value of the counter is compared to a predefined threshold. If the counter is below the threshold, the counter is incremented and the packet is admitted to the network. If the counter is equal to or above the threshold, the packet is dropped. The counter is decremented at a predefined rate as long as it is positive [37]. This rate shaper is used in a control loop, similar to the the methods in which basic learning methods would institute control over senders.

CHAPTER 6
Parameter Optimization

All network, system, or application managers have a configuration which may be changed to alter the behavior of the system. Many times, the optimum configuration is unknown and administrators constantly twist many knobs of the system in search of the best performance. To complicate matters, in most cases the optimum configuration would change as temporal, application, and user behavioral patterns change. In this sense, the optimum configuration may actually be a function dependent upon several factors (including time) making the maintenance of a constant optimum configuration virtually impossible for a human administrator.

In this chapter we will first mention some of the current corporate efforts towards system and network parameter configuration and optimization. We will then discuss our network emulator using neural networks, followed by the description of other research efforts in this area.

6.1 Current Standards

The typical method of modeling or understanding computer systems and networks is the administrator learning the behavior of the hardware during the normal repeated uses of the organization's everyday operations. Many of the diagnostics and management applications on the market give the administrator a great deal of information about the operations, preparing the administrator to make educated decisions regarding optimizing its operation. However, many times the administrator is so busy fighting problems in the network or with the systems that he or she does not get around to optimizing the network with this information.

A few companies are moving towards creating products which can configure and optimize their own behavior on the fly [80, 127]. IBM is one of the companies leading the pack in this effort and has coined the term referring to this capability: "*Autonomic Computing.*" Some early corporate efforts illustrating some of this capability include: Microsoft's AutoAdmin project [31], IBM's eLiza [180], Hewlett-

Packard's Utility Data Center [94], Sun's N1 project [140], Apple's Rendezvous technology [71], and Sychron's Application-Resource Brokering technology [156].

6.2 Neural Modeling for Network Parameter Optimization

The systems we would like to optimize have many autonomous components and complicated data flows. Many of these system cannot be modeled easily through traditional methods. In this section we will discuss a modeling technique for use in optimizing parameters in computer networks. An optimization process should be able to use our model to predict what benefit or cost a particular configuration offers.

Network optimization is an issue in many networks which need optimal restraints placed on parts of the network to adhere to an agreed upon level of service. A wide variety of network parameters can be optimized including control parameters for participating nodes, active queue (RED, FRED etc.) parameters, routing protocol parameters, and any other group of parameters that can be set by an administrator.

Theoretical models could be used for this purpose, but many of these ignore or disregard some of the detailed characteristics of the system. It may be a combination of these detailed characteristics that achieve the optimum performance. Testing new parameters on the actual network would not be effective because the outcome is unknown. To provide a more comprehensive coverage of all parameters during modeling, we use simulation.

There are two big problems with using network simulation as a part of any network management package. The first problem is the simulator's accuracy and relationship to the real data network. If the simulator is not very accurate as to the operations of a real network, then it would be of no use to any network monitoring application. The second problem with using a network simulator as part of a package is speed. The simulator can take so long to simulate, that the new solution is no longer relevant to problem simulated. To be a part of real-time applications, the simulator would have to be very fast and quite accurate.

To address the aforementioned problems, we use a neural network to emulate

the behavior of the network simulator. In our learning phase, we train the neural network to give us only the statistical values we want to know. To ensure accuracy, the neural network was trained on an extensive data set. Once the statistical behavior of the network simulator is learned, immediate answers can be given from the neural network.

6.2.1 Simulation Emulation

We used *NS*, a discrete-event network simulator targeted at networking research, to model the network and different scenarios of network traffic [12]. We used the same network depicted in Figure 5.2 with the same parameters, like packet size, link delay and bandwidth as described in Section 5.2.1.1.

There are many ways in which one could claim the network must excel to be considered the "more efficient" network. To be compatible with existing standards, and remain easily deployable on a real network, our prototype considers statistics that could be determined using the well-known SNMP protocol (Simple Network Management Protocol) and the variables defined in MIB file RFC1213 [115]. In this case, our model will predict the number of bytes delivered (SNMP variables *ipInDelivers* and *ipForwDatagrams*) and the number of bytes dropped by the network (SNMP variables *ipInDiscards* and *ipOutDiscards*). The adjustable parameters of our prototype were bandwidth limits placed on each host. In the simulator this was simply an adjustable parameter, whereas in a real network it could be done with ipshapers, or other control techniques (Recall the discussion of current control methods in Section 5.3.3).

The parameters we are predicting, *ipInDelivers*, *ipForwDatagrams*, *ipInDiscards*, and *ipOutDiscards*, are useful parameters to optimize, especially in environments where packet drops and retransmissions are costly. These types of conditions are prevalent in satellite communications and other wireless communications where the normal communication protocols do not lead to optimal results. However, other optimization parameters and control variables could be used in a similar fashion.

6.2.2 Training the Neural Network

The goal of the neural network is to emulate the network simulator, by returning the number of bytes delivered and the number of bytes dropped by the network under varying conditions. After training, the neural network would be able to return this information immediately while the simulator would take time to run. The neural networks used for this purpose have an initial input layer which takes each network node's bandwidth limit as input. Based on this set of bandwidth limits, the neural network outputs both the number of bytes delivered and the number of bytes dropped. A visual explanation of the neural network structure is given in Figure 6.1.

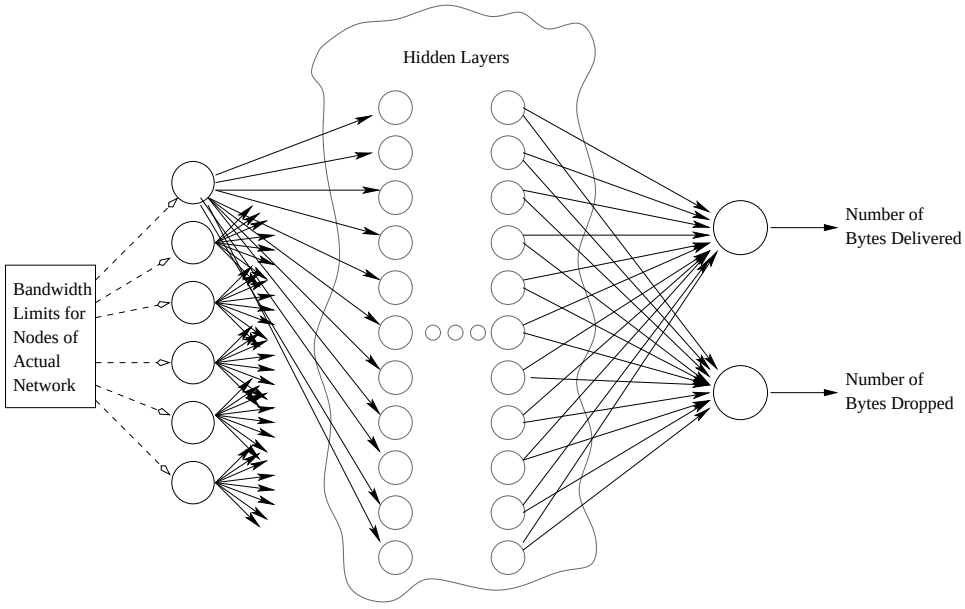


Figure 6.1: Neural Network Configuration

Several neural network architectures were attempted to discover the best number of layers and neurons per layer. The first network used had a single hidden layer of 10 nodes. This neural network was trained with about 30 training samples. Unfortunately, this neural network would not emulate even the most restricted data network. There needed to be many more training samples to give the neural network learning experience in the many areas of the network simulator.

To address this issue, a data collection program was written to retrieve sev-

eral thousand samples uniformly distributed over the set of networking bandwidth limits in a particularly large range. This program ran the simulator over several exaggerated time periods for each sample and recorded the values. The reason these experiments are said to be exaggerated is because the experiments were conducted with the simulator running several times faster than real time to reduce the amount of physical time necessary for simulation (i.e. for smaller networks a discrete event simulator, such as NS, may be able to simulate 5 real-time minutes, in only 1-2 minutes). This program created 15,625 samples after 9 hours. This recorded data was then prepared for learning by an additional program which randomly selected the requested number of cases for learning and testing.

After this procedure was created, training sizes of around two hundred were used. These changes yielded better, but still unsatisfactory results. The best results were achieved when the neural networks were trained and tested using all samples (15,525 for learning and 100 for testing). The results in these cases were much better, however, they took a great deal of time to train (around 30-48 hours).

We tried training neural networks with different numbers of hidden layers to further determine the best neural network architecture. A small graph of the squared error with respect to the number of hidden layers is given in Figure 6.2. As can

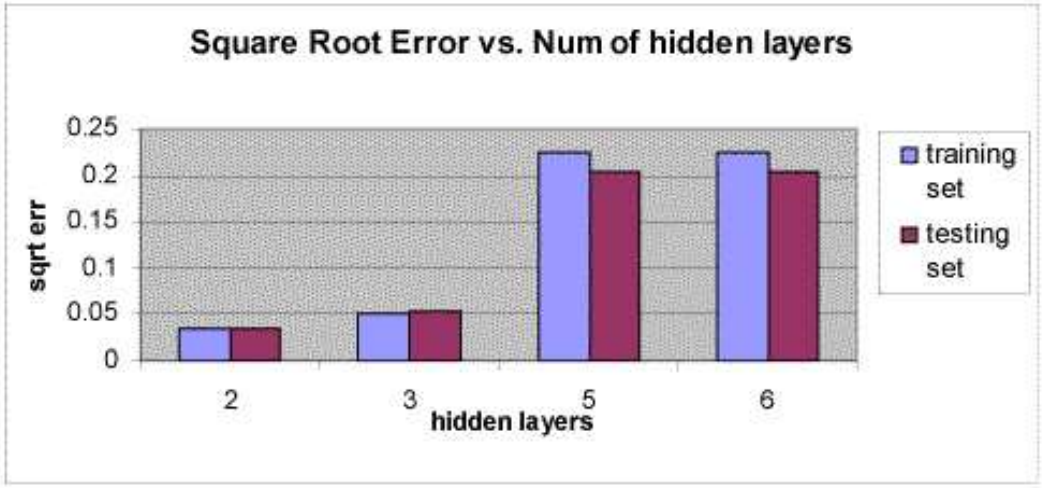


Figure 6.2: Training Error vs. Number of Hidden Layers used

be seen, the smaller the number of layers, the better the neural network did. The

networks with a small number of layers were able to train much faster, and therefore were able to train over more iterations than those with more layers. So these results were a little unfair. However, the two cases in which there were 5 and 6 hidden layers were a little disappointing. After a couple of thousand iterations (around 20 hours), their error rate was expected to drop, but they remained the same. These two cases were expected to converge slowly, but not to stand still. Initially, we tried these because we thought that we may be experiencing the phenomenon of the hyperplane, a linear separable problem, in which the addition of certain nodes of the data network would require a different hidden layer in the neural network for effective emulation. However, once again, this may be a factor of the number of training samples and the number of weights involved in such a large network. Incidentally, Kolmogorov proved in the 1950's that the correct number of hidden layers is always two or less [102].

6.2.3 Testing the Trained Networks

Scatter plots of the first 30 points in the test sets of 100 are shown for the number of bytes received in Figures 6.3 and 6.4. To simplify viewing, we did not show all 100 points. Plots were not given for the networks with 5 and 6 hidden layers because their error rates were too high (see earlier bar graph in Figure 6.2). The most difficult network characteristic to predict was usually the number of dropped bytes. This is partially due to the large number of zeros in the data set. Similar plots for the predicted and actual bytes dropped are shown in Figures 6.5 and 6.6.

As seen in the scatterplots, the neural network emulators trained with 2 and 3 hidden layers give near-perfect values for network simulation. More work can be done to fine tune these predictions. Also, it may be fruitful to split up the prediction of bytes delivered and the bytes dropped into two separate neural networks. This is a well known way of improving the performance of neural networks having more than one output.

6.2.4 Summary

In this chapter we have shown how a neural network can be used to model a limited set of network behavior. We have illustrated the model's accuracy through

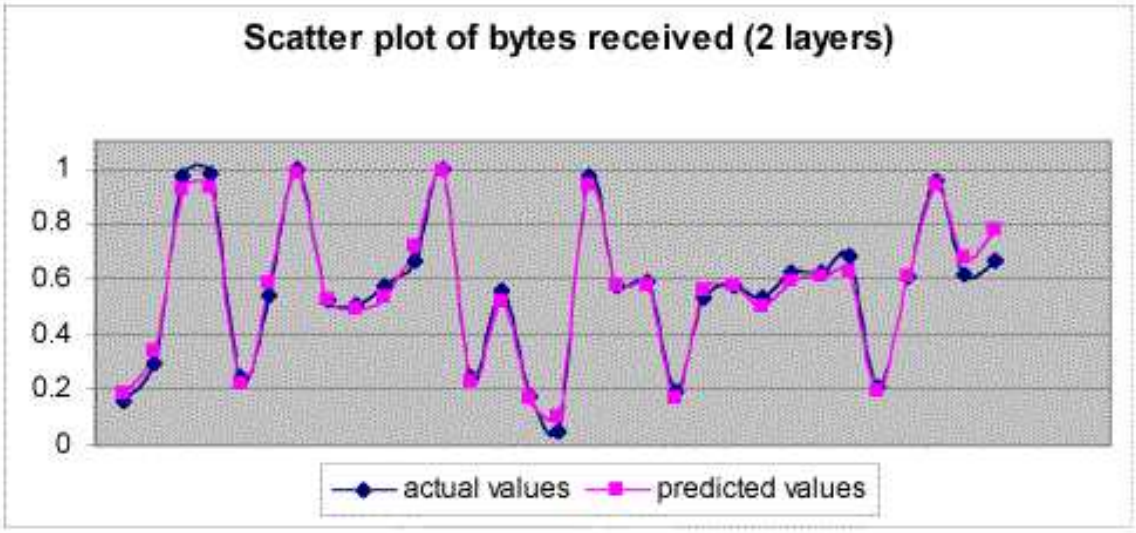


Figure 6.3: Bytes received Scatterplot from Neural Network with 2 hidden layers. The y axis contains normalized values of bytes received while each point on the x axis represents a set of parameters given to both the network simulator emulator and the actual simulator

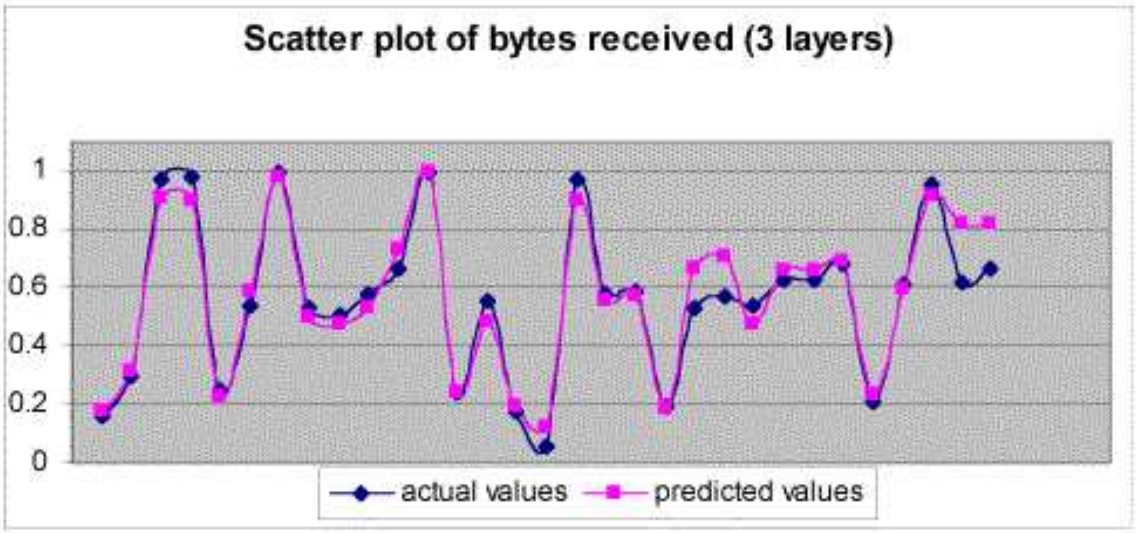


Figure 6.4: Bytes received Scatterplot from Neural Network with 3 hidden layers. The y axis contains normalized values of bytes received while each point on the x axis represents a set of parameters given to both the network simulator emulator and the actual simulator

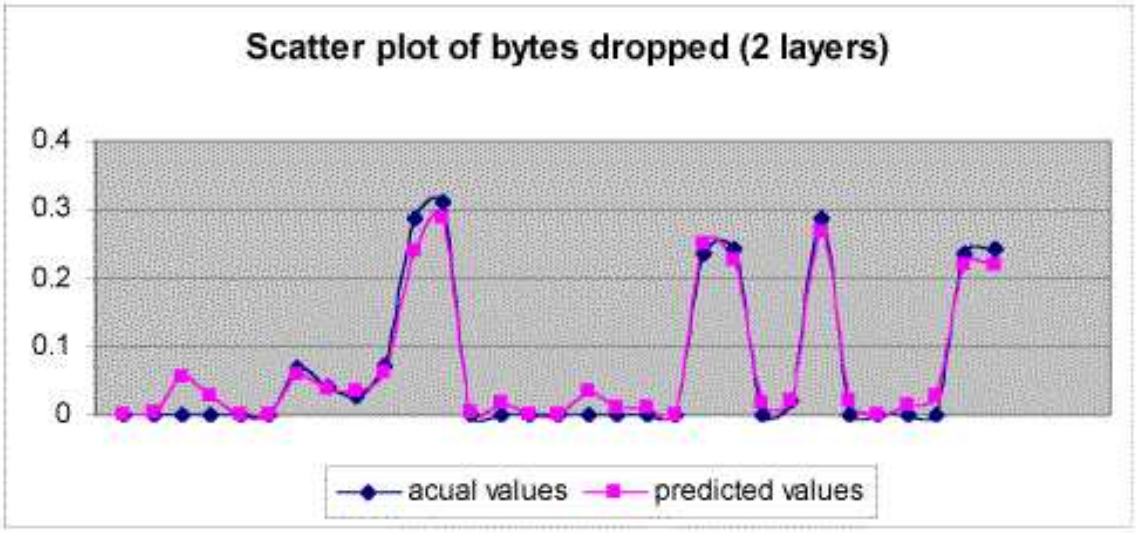


Figure 6.5: Bytes dropped Scatterplot from Neural Network with 2 hidden layers. The y axis contains normalized values of bytes dropped while each point on the x axis represents a set of parameters given to both the network simulator emulator and the actual simulator

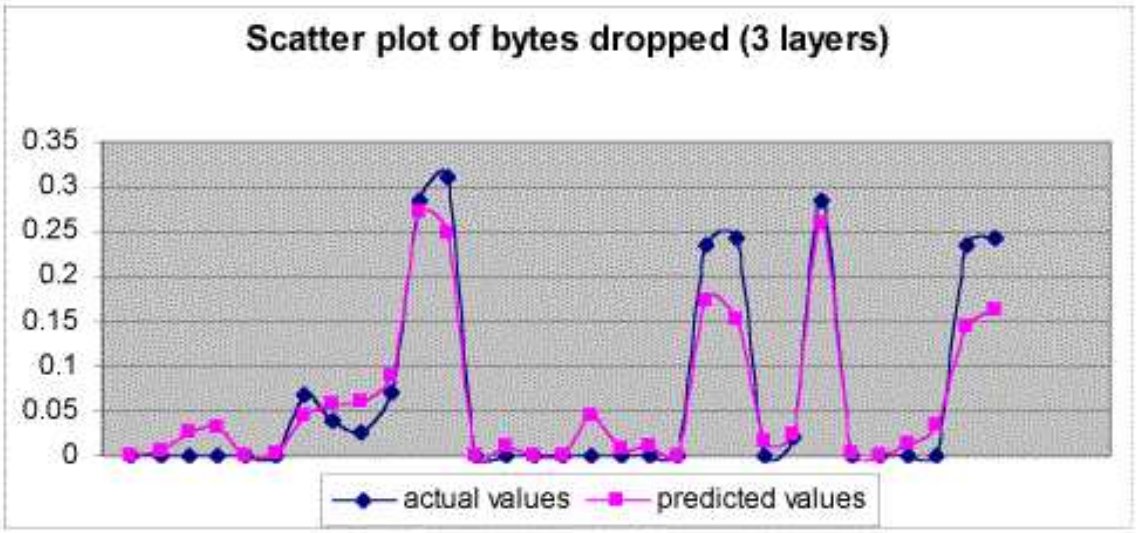


Figure 6.6: Bytes dropped Scatterplot from Neural Network with 3 hidden layers. The y axis contains normalized values of bytes dropped while each point on the x axis represents a set of parameters given to both the network simulator emulator and the actual simulator

scatterplots of portions of the dataset. Once such a model is created, it can be used as a component of a self-aware or self-optimization technique (described in Section 7.2.1.3). We feel that neural models could be developed to model other network behavior as well as other complex systems.

6.3 Related Works

Network parameter optimization remains one of the challenges of network research. Because all groups do not have the capital or other means to assemble a private lab for network research, simulators are many times used as testing grounds for new techniques. As simulators become faster, more accurate, and more scalable, researchers are considering using these tools online, just as we have, bundled as part of a network management application.

A group in Rensselaer Polytechnic Institute created a special network simulator based on NS *(network simulator)* to be used online in network management applications [182, 187]. This network simulator was essentially a distributed version of NS to execute several single machine experiments across several workstations. The creators used a farmer-worker architecture with an added topological decomposition procedure to speed up normal simulations.

Another innovation is the Genesis system based on a novel approach to scalability and efficiency of parallel network monitoring, modeling and simulation [161, 162]. This approach is based on network decomposition that creates separate network domains. Each domain is independently monitored, modeled and simulated by separate software components and using fast traffic generators [185]. One of the desirable features of the design is its independence from the underlying simulators and repositories running in the individual domains. Hence, this architecture was integrated with a number of existing technologies thereby supporting system interoperability [160, 159].

This group designed also a search algorithm based on the hill-climbing and TABU techniques to find increasingly better sets of networking parameters (placing the parameters in their distributed network simulator to evaluate each parameter set). Using this technique, they optimized RED parameters in a real network, and

proposed a plan for optimization of OSPF routing parameters.

The "Hindsight Optimization" group at Purdue University uses traces of potential future network behavior to evaluate possible network control actions with the end goal of optimizing the state of the network. In hindsight optimization, the "utility" achievable from a particular state of the system is estimated by averaging values obtained from a number of traces starting at that state. In other words, to evaluate the benefit of the network being in a certain state they will assume that state and carry out a great deal of simulations from there. Once many of these simulations have been performed for a particular state, the average benefit of those simulations is calculated. Given a set of states from which a network state should be chosen, the algorithm should pick the state with the highest average benefit from its possible future states. This gives the controller the benefit of a relative 'hindsight" of its actions before it takes action. A simulator is used to obtain the many utility values, so like other optimization strategies, the simulator must be fast. To speed up their simulations, they only retrieve relative values from the simulator. The authors claim that this relative simulation saves them time and does not affect the results of the optimization because the same relative error is present uniformly across the system. They used this strategy to optimize multi-class scheduling, where the objective is to minimize the weighted loss of packets. They also used this strategy for congestion control, where the objective is to improve utilization, delay, and packet loss while maintaining fairness in a network with a bottleneck and fully controllable sending nodes [32].

Researchers from the National Institute of Standards and Technology build networking models using a different form of soft computing called cellular automata [184]. In this work, they model a network as a two-dimensional cellular automaton to gain a greater understanding of the spatial-temporal evolution of network congestion, and other emergent communication network phenomena. They use this model to study dynamic patterns arising from interactions among traffic flows routed across shared network nodes while using various configurations of parameters and two different congestion control algorithms. They too, acknowledge the ability for simulation to render useful and accurate metrics for these studies, but similar to

our work, needed a faster solution.

CHAPTER 7
Summary and Future Works

7.1 Summary

This thesis has examined many avenues of efficient, scalable network management. This section summarizes the contributions of this work.

In ***Chapter 2*** we described frameworks for network management including our agent-based distributed architecture. We also describe industry standards for network management and how they compare to our framework in their effects on the managed network and client applications. We provide a thorough analysis of our framework through extensive laboratory and Internet-based experiments including the following scenarios and varying parameters:

- Small (three router) network
  - traditional data collection
  - preprocessing collection
- Autonomous Systems Topology
  - traditional data collection
  - preprocessing collection
  - background UDP and ICMP traffic
- Two networks connected by the Internet

Through simulation we also extended our evaluation to include the following topologies and traffic variations:

- Autonomous Systems with tftp background traffic
- United States Internet Topology with tftp background traffic

In order to evaluate the complicated behavior of our framework through simulation, we developed a separation technique by which we segment the many flows of a particular application into easily simulated, separable flows. A formal analysis and extensive explanation of such a separation, including flow requirements, limitations, and presence, are given in ***Appendix C***. This type of flow separation now makes it possible to evaluate the effectiveness of certain application frameworks within certain error bounds.

Results from evaluating the DOORS framework against the traditional SNMP solution yielded the following:

- In a single-client scenario, we see a cost benefit of running DOORS for monitoring network data as compared to conventional SNMP polling methods.
- In a multi-client scenario, DOORS outperforms standard polling methods and this difference grows linearly as a function of the number of clients polling for similar data. DOORS achieves this advantage thanks to the consolidation of multiple client requests into a single aggregated request.
- DOORS uses TCP connections which make the data transfer inherently reliable compared to standard SNMP polling methods which use UDP. The added functionality of TCP, comes at the cost of extra bandwidth in the form of added transport layer headers. However, DOORS counteracts this cost by reducing the total number of data messages which pass around in conventional network polling.
- DOORS has proven useful in cases where normal SNMP polling is not feasible, and the management application has no control over the networks in-route to the managed networks.
- DOORS can be extremely effective when encoded with functionality beyond just the simple collection and return of data. When some or all of the algorithm from the client is placed into the agent, we can see large savings on bandwidth and speed of calculation.

- Using DOORS, the client will get its data faster, but may have small deviations in the time difference between polls, whereas normal SNMP clients will get the data later than the DOORS clients, but at a more consistent inter-polling interval.

The DOORS system is an efficient way to extend any network management application in a scalable, easily-managed fashion. To show the applicability of the DOORS framework in the different areas of network management, we have developed several network managers and shown how they may be distributed within the DOORS framework. The remaining chapters describe these managers and how they each contribute to their respective areas.

In ***Chapter 3***, we focus on network-based intrusion detection and review two IDS managers that we have created. The first of our network-based managers uses a time-dependent deterministic finite automata to define attack signatures in tcpdump data. The second manager that we created uses a self-organizing map for clustering source behaviors and an Perceptron-based artificial neural network to label the collection of behaviors as an attack or normal traffic. SOMs are frequently used as visual clustering techniques, but we needed them to function as automated clustering agents. For this reason, we extended the traditional SOM structure to keep various statistics in the SOM learning phase. We then developed a specialized automated clustering algorithm for SOMs known as *"Frequency-Based Clustering"* described in detail in ***Appendix D***. ***Chapter 3*** also briefly discusses some of the corporate efforts for network-based intrusion detection and explains, in a bit more detail, other research efforts in this area.

Chapter 4 discusses efforts in host-based intrusion detection, including our probabilistic state host-based masquerade detection system. Our system uses this special type of automata to describe user signatures, in which likelihood values are given to each command, to compare with future user activity. Careful measures are taken not to punish advanced users with an extensive command library (very wide automata structures) and/or fast typists (very long automata structures). The structure of the automata in this approach is based on the time difference between command entries, a quantity ignored by other masquerade detection research efforts.

We explain other research efforts and briefly review some of the corporate products available for host-based intrusion detection.

We review approaches aimed towards congestion control in data networks in ***Chapter 5***. We describe the industry standard ways of handling congestion, along with some research efforts to predict and prevent congestion. In this chapter we also present our network congestion control detection manager which uses neural networks to predict congestion and its sources. Once congestion is predicted and the sources found, we take measures to reduce the sending limits of these sources. Our results show that we were able to predict the congestion and stop it before packet loss in several cases. Other stronger cases of congestion were able to be mitigated soon after packet loss.

Chapter 6 presents our method for modeling data communication networks for use in a predictive capacity in network management. We present this work as a component of a network parameter optimization module, as it can quickly return the effects of perturbations of various parameters in a data network. One method of achieving these type of results is from theoretical models, which frequently ignore subtle changes in collections of small parameters. Simulators and live networks can also be used, but take a great deal of time to represent large scale systems. Our method uses a neural network to emulate a network simulator, returning the same results that the simulator would, but returning the results instantly once the neural network is trained.

In the next section, the Future Works section, we talk about the many directions our research can be extended. We also devote a particular subsection to providing an outline for each of the network managers discussed to be functionally distributed using a framework such as the DOORS system. This functional distribution provides each of the methods with ways of reducing loads on management hosts, deploying and distributing updated data and management structures, and avoiding both hardware and protocol limitations. As networks continue to be used for new applications, the ability for management applications to keep up with this growth becomes increasingly important. When scalability, reliability, and efficiency are the questions – a distributed framework for management applications, like DOORS, can

be the answer.

7.2 Future Works

Our work has been extensive, but there are many directions that we would still like to pursue. This section will briefly introduce some of these directions and why they would be beneficial to pursue. Secondly, we also devote the last half of this section to discussing how many of the network managers mentioned can be functionally distributed with a framework like the DOORS system.

7.2.1 Extending Current Methods

7.2.1.1 DOORS

To conclude our analysis of the DOORS system, it would be prudent to evaluate exactly when the benefits of systems like DOORS start to break down. After all, there is a set of parameters (e.g. topology configurations, background traffic, and management component placement) that will expose the worst in any system. We are in the process of analyzing the effect of DOORS when various distances are imposed between DOORS components. We are using two of the distance metrics recommended in [81] and our own additional metric, the number of background traffic sources, to form one composite distance metric. We may then use these results to determine when the application of a system like the DOORS system adds no additional benefit. Other interesting studies would compare DOORS with other traditional network management systems besides SNMP, for example with RMON or CMOT.

7.2.1.2 Intrusion Detection

In our intrusion detection work, an interesting extension to the matching algorithms used to determine if signatures are followed could be found in the area of bioinformatics. In particular, alignment algorithms such as the Smith and Waterman algorithm[146], could give good results as they are made to recognize similarity in sequences which may be interrupted by smaller sequences of junk data. This condition precisely describes the state of the masquerading datasets and could also

be used in some misuse detection systems for both network-based and host-based datasets.

We would also like to pursue the the next natural progression to intrusion detection, intrusion prevention [38, 39]. Currently, our IDS prototypes have only the ability to detect an attack, but we could institute stronger control or preventive measures as our detection algorithms notice stronger signs of danger. Lastly we would like to have more extensive datasets to use while evaluating our systems. To this end, we have started efforts to generate our own controlled datasets by organizing local user sessions and creating our own attack library.

7.2.1.3 Parameter Optimization and Neural Modeling

As we have shown, modeling data networks with artificial neural networks is an effective way to calculate the effect of changing parameters in the system. The speed of this approach could enable us to use many algorithms that need a predictive module to quantify such effects (illustrated in Figure 7.1). In this section we will briefly discuss the implications this technique could have on autonomic computing, and how it could be used with traditionally slower algorithms to achieve speedy evolution of optimized parameters.

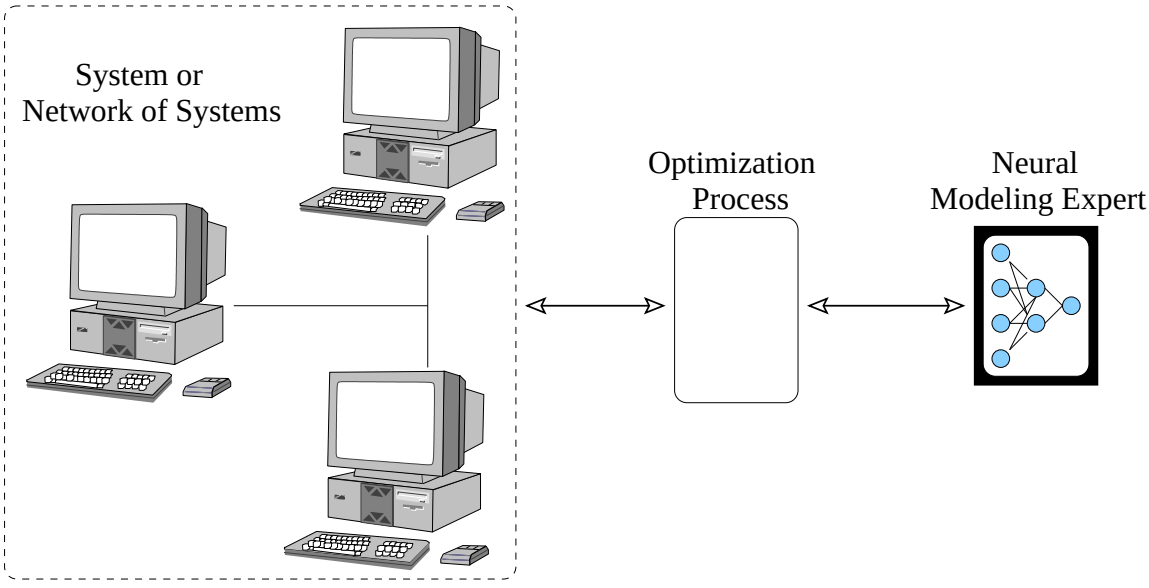


Figure 7.1: General Neural Model Architectures

Autonomic Computing

One particular aspect of autonomic computing that lends itself to this type of modeling is parameter and system optimization. For the system to understand how to improve itself, it must have a concept of its current condition. This concept of self state is what a neural model can provide. It can learn from past behaviors, and possibly use extra cycles to test various loads to build a comprehensive set of data from which to learn. A neural model such as the one we have described can also provide the system with the ability to get an estimate of what the current perturbation of system parameters will do to the system. This would be useful in admission control and self-optimization systems.

Genetic Evolution of Network Resource Optimization

The benefits of a neural model emulating a network simulator can facilitate the real-time use of traditionally slower optimization techniques (genetic algorithms, hill-climber methods, etc.). In this section we will show how a genetic algorithm can be used for network parameter optimization using the neural model discussed earlier.

The use of genetic algorithms to evolve optimal networking parameters for data communication networks has been tried in the past [69, 18], but abandoned for several reasons. The first reason is that genetic algorithms use a cost/benefit function, which requires an accurate cost/benefit value associated with each set of networking parameters (See [68] for detailed discussion of Genetic Algorithms). This is a non-trivial process, because the effects of traffic in a data communications network can be difficult to model. Typically, network simulators are used to evaluate these effects by simulating interactions of each packet with other packets and networking equipment. However, simulators are usually slow. This leads to the second reason why the use of genetic algorithms in network optimization has been abandoned. Genetic algorithms test many parameter sets to evolve the strongest of the population, so they are usually considered slow to find solutions. Couple this slow process of finding solutions with the slowness of a network simulator to determine

cost/benefit metrics and the result is an extremely slow optimization process. If the goal was real-time optimization, by the time the genetic algorithm evaluated the cost of members in its population, the conditions of the evaluation may change.

To combat the length of time taken by the cost function, we train a neural network to act as a network simulator. This allows us to train the neural network once, and use it as a simulator for each member of the population through each generation. Once the neural network has been trained, the time taken to evaluate new behavior based on several traits is minimal.

If we were to design a genetic algorithm to adjust bandwidth limits to optimize the number of bytes received in the network and the number of bytes dropped by the network, we could use the neural model described earlier in the thesis. In this genetic algorithm, a fitness function must be created to determine the health of the network with a certain set of parameters (hard limits). This function must be calculated to determine "network fitness" for each member in our population of solutions for the genetic algorithm. An example fitness function can be found in Equation 7.1.

$$Fitness = \quad DataBytesReceived * C - BytesDropped \tag{7.1}$$
$$where\ C = \frac{C_1}{C_2}$$

In the above fitness function, C_1 is a constant specifically used to bias the data bytes received. C_2 is used to set the bias for the number of bytes dropped. The constant C_2 can also be interpreted as a penalty factor. Many times, the number of packets dropped will be much smaller than the number of bytes delivered.

A system such as this can evolve networking parameters in networking environments. The genetic algorithm process may still take some time to run through its many generations; however, it would realize a speedup proportional to its population size by using our neural model.

7.2.2 Functional Distribution of Network Management with DOORS

In this subsection, we focus on extending the intrusion detection and congestion control managers using the DOORS system.

7.2.2.1 Distributing Intrusion Detection Managers

As networks grow in size, speed, and segmentation, scalability problems continue to plague many network-based intrusion detection systems. Recent surveys have shown that current network-based IDSs have trouble keeping up with modern day traffic [122]. In gigabit networks, packet loss and other failures are not uncommon with even high-speed intrusion-detection systems, which have to duplicate traffic to analyze it [119]. It has also been noted in the network security field that the growth of switched and other highly segmented networks has posed a significant problem for current intrusion detection methods which use sniffed data for detection [117]. A recent report by TopLayer's Simon Edwards points out that there are also several hardware and network configuration issues that prevent intrusion detection systems built to operate at the appropriate speeds in segmented environments from getting all of the necessary data [52]. Scalability concerns are also present in host-based intrusion detection systems.

A logical extension of the intrusion detection architectures mentioned in Chapter 3 and Chapter 4 is to distribute the detection functionality. This functional distribution will serve to reduce the collection load per machine, pushing the detection deeper in the network. Projects such as EMERALD use a similar scalable distribution of surveillance monitors throughout a network to apply distributed event correlation models in detecting network intrusions [128]. Our intent is to use the DOORS system in which our detection process would be encoded into mobile agents and sent to the network in need of monitoring [20]. An illustration of this type of distribution pushing the detection functionality deeper into the network can be found in Figure 7.2 (please recall previous illustrations of traditional architectures from Figure 3.1 for comparison).

To facilitate distributed detection, it is important to recall the modular framework of applications reviewed in this thesis (Figures 3.5, 3.8, and 4.2). As illustrated in Figure 3.5, the SSO, or a client, through the DOORS repository, can update the IDS agent by sending new TDFAs to the TDFA provider of the IDS agent. Administrators have similar abilities for the neural network detection manager (shown in Figure 3.8), as updates to both the learned SOM structure and MLP weight struc-

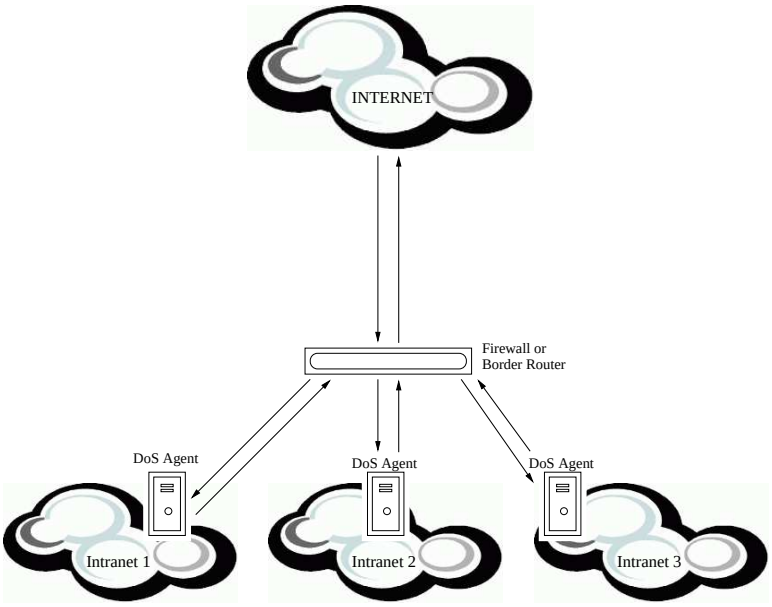


Figure 7.2: Distributed (DOORS) Network-Based Intrusion Detection

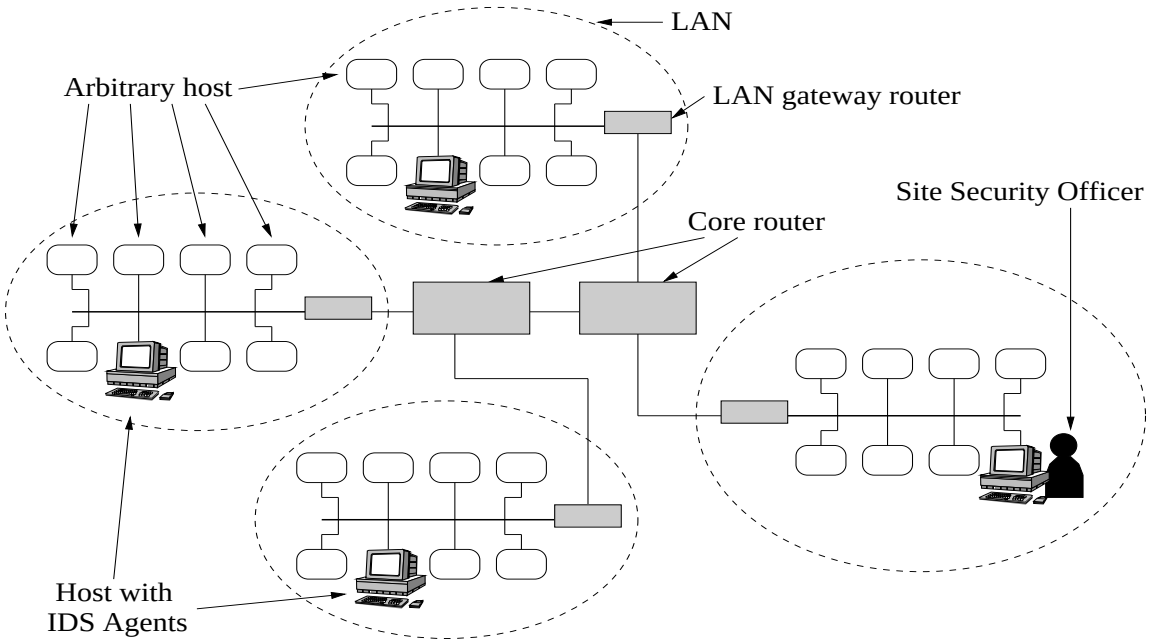


Figure 7.3: Overview of distributed modular architecture

ture can be provided through the NN structure provider module. Lastly, Figure 4.2 shows how new PSFA structures can easily be added to the system through the PSFA provider.

Figure 7.3 illustrates the topology of such a network middleware architecture. This figure is a portrayal of a large network in which smaller LANs are connected by gateway and core routers. In this case, a host in each monitored network will be equipped with an agent server to receive an agent containing the appropriate IDS code. A site security officer will update the host running the IDS with new learned structures when necessary through repository to agent communication. We believe that this solution would effectively distribute the detection while providing a centralized management for continual updates.

7.2.2.2 Distributing Congestion Control Managers

A slightly different extended architecture can be formed for the congestion control manager by dividing its larger networks into domains small enough to influence easily (most large networks already contain these divisions). When this is in place, a separate protocol or addition to current protocols can be used to determine domain-to-domain agreements.

As illustrated in Figure 7.4, a learning control agent would be somewhere within each domain defined in the network. Using the forecasting and detective powers of the agent, each domain would be regulated and operating at safe levels. To negotiate any problems between domains, the control agents will communicate with each other as to the effects of one domain on another. The control agents will take this information into account just as it took the information from local nodes into account. This will promote the same safe state of operations between domains as well as inside of each domain.

In this case, DOORS agents can be equipped with managers capable of communication through agent-to-agent communication. Using the DOORS system we can send the trained agents out to the domains to police the individual networks while training new agents to replace the old. This would provide a constant updating strategy that our management technique can use to adapt to continual changes

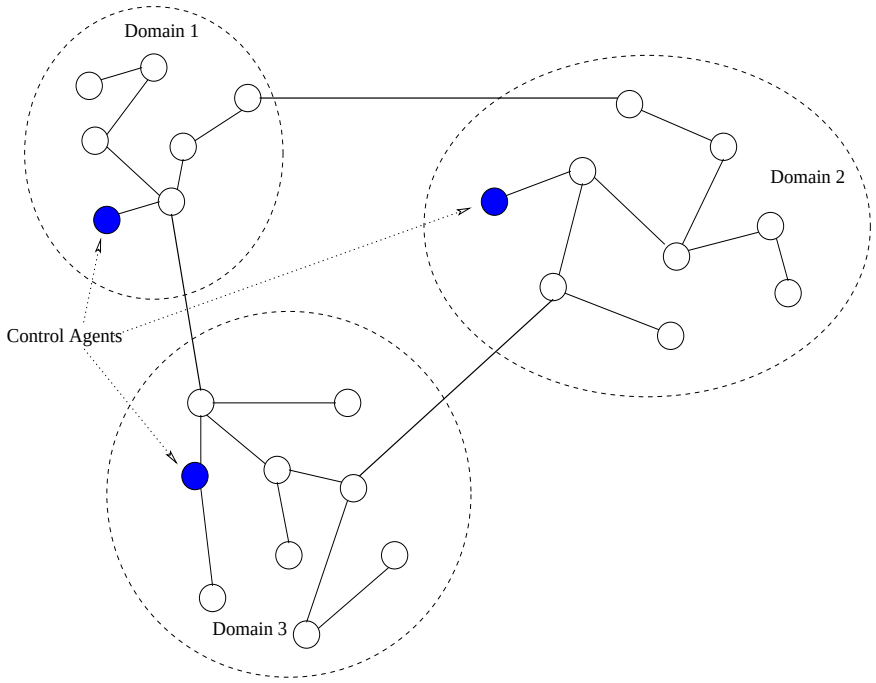


Figure 7.4: Extension of the architecture

in network flow patterns.

APPENDIX A
TCP Congestion Control

TCP (Transmission Control Protocol) uses a combination of several algorithms to control congestion. We will first look at basics of TCP and then examine two of the algorithms used to improve it [150, 151, 34]. TCP is used to provide reliable transfers of information between computers connected to each other through a network. When a TCP connection is established, the sender and receiver exchange messages to agree on the best sized chunks, called *segments* (measured in bytes), to use for transmission. These chunks will be sent to the IP (Internet Protocol) layer of each machine for a transmission. When TCP sends a segment, it maintains a timer while waiting for the other end to acknowledge the reception of the segment. If an acknowledgment (ACK) is not received in time, a loss event has occurred, and the segment will be retransmitted. If the segment is acknowledged in time, the next segment is sent.

To speed up this process, TCP uses a windowing technique. Instead of sending one segment, and waiting for the response before sending the next segment, TCP sends a window of segments. Once the window of segments is sent, the sender waits for the receiver's ACK of the segments sent. The window size (measured in bytes) used in a TCP connection is also negotiated by the sender and receiver at the time the connection is established.

Because of the many negotiations taking place in the beginning of a TCP connection, an example may be helpful:

> If machine A would like to develop a TCP connection with machine B, A sends a request for connection as well as, amongst other items, a proposed maximum segment size and proposed sending window size. The receiver acknowledges this request for connection and gives its maximum segment size and the size of its receiving window. The minimum of the two maximum segment sizes is used as the segment size of the connection. In turn, the minimum window size between the sender's sending window

and the receiver's receiving window is used as the window size of the connection (the lower value is usually the receiving window size so we will refer to this variable as *recwnd*). This value may change at various times during the connection as the receiver's buffer is filled and emptied.

In the first implementations of TCP, after initializing the connection, a sender would start transferring data by sending as many segments as it could fit into the agreed upon window size *recwnd*. For local networks, this may have been efficient, but for networks which spanned across several routers and other networking equipment, it quickly lead to congestion and router buffer exhaustion.

The algorithm that TCP uses to avoid this effect is called "slow start". Slow start adds a variable called the congestion window (*cwnd*, also measured in bytes) to TCP senders. When a new connection is established with a host, *cwnd* is initialized to one segment. The *cwnd* is what the sender uses to determine how many segments it may send before waiting for an ACK. The maximum value of *cwnd* is the *recwnd*. Once the first segment is acknowledged, the congestion window is increased by one segment. The congestion window serves as a means of flow control imposed by the sender, while the *recwnd* enacts flow control imposed by the receiver. The former is based on the sender's assessment of network congestion, while the latter is usually related to the amount of available buffer space that the receiver has for this connection.

Because the sender is sending multiple segments before waiting for an ACK, the receiver must now identify which segment it is acknowledging when replying to the client. However, the receiver knows that the sender will be sending a window of segments, so instead of acknowledging every segment, it may wait to acknowledge multiple segments at once. When the receiver chooses to send an acknowledgment, it simply sends the sequence number expected next (even when the segments arrive out of sequence). This means that when the receiver receives a segment with a sequence number of 5, and it expected a sequence number of 3, it will send an acknowledgment containing the sequence number of 3 (which may be a duplicate acknowledgment, because the receiver may have acknowledged the segment with sequence number 2 with the same ACK). Therefore, a sender receiving several du-

plicate acknowledgments (usually 3) is also an indication of a loss event.

After establishing the connection, the sender starts data transfer by transmitting one segment and waiting for an ACK. When that ACK is received, the congestion window is incremented from one segment to two segments. Now the sender can send two segments before waiting for an acknowledgment. When each of those two segments is acknowledged, *cwnd* is increased to the size of four segments. This provides close to exponential growth, which makes the name "slow start" an interesting one. However, compared to the sender just transmitting the receiver's *recwnd* at the beginning of the connection, the method is truly a slow start.

At this point, we have only discussed how successful acknowledgments effect the *cwnd*. If a loss event occurs, either through a timeout or via 3 duplicate acknowledgments, the sender thinks it has sent too much at one time (its *cwnd* has gotten too large) and prepares to switch from slow start to congestion avoidance. Congestion avoidance (sometimes referred to as additive increase/multiplicative decrease) is an algorithm that TCP uses to avoid further congestion in the network. It adds another variable, the slow start threshold (*ssthresh*) measured in bytes, to preserve the last *cwnd* value before congestion occurred. When the loss event occurs, *ssthresh* is set to one-half of *cwnd* ("multiplicative decrease") and *cwnd* is reset to one segment (1 MSS, Maximum Segment Size). The variable *ssthresh* serves as a limit for the slow-start algorithm to stop experiencing exponential growth, and signals the switch from slow start to congestion avoidance. Therefore, to determine which mode TCP is in, you must examine the values of *cwnd* and *ssthresh*. If $cwnd < ssthresh$, TCP is in slow start; however, if $cwnd >= ssthresh$, TCP has switched to congestion avoidance mode.

Congestion avoidance dictates that once $cwnd >= ssthresh$, *cwnd* should be incremented by $\frac{segment\_size * segment\_size}{cwnd}$ each time a segment is acknowledged. Therefore, if all of the segments in the *cwnd* are acknowledged, *cwnd* is only increased by one segment. This is a linear growth of *cwnd*, compared to slow start's exponential growth (doubling *cwnd* when all acknowledgments in the window are received). An example of *cwnd* during congestion avoidance can be found below:

To reduce the formula for the increase of *cwnd* in congestion avoidance

mode, $\frac{segment\_size*segment\_size}{cwnd}$, we will introduce the formula for the number of segments (*num_of_segs*) in the *cwnd*. Recall that the *cwnd* is in bytes, so the number of segments (*num_of_segs*) that can be in the *cwnd* are $\frac{cwnd}{segment\_size}$. Using substitution, our new formula for the increase of *cwnd* in congestion avoidance mode is now $\frac{segment\_size*1}{num\_of\_segs}$.

If *cwnd* is the size of 4 segments and the TCP connection is in congestion avoidance mode, all 4 segments are sent to the receiver. When a segment is acknowledged, *cwnd* is increased by $\frac{segment\_size*1}{num\_of\_segs}$, where *num_of_segs* is 4. To simplify, the *cwnd* is increased by $\frac{segment\_size}{4}$. By the time all four acknowledgments come back, this process would have happened four times increasing *cwnd* by $\frac{4*segment\_size}{4}$ or one segment_size.

The linear growth of *cwnd* is where the "additive increase" part of the algorithm comes in. It has been shown through research that the additive increase strategy is a "necessary condition for a congestion control mechanism to be stable [4]."

The behavior explained here describes TCP Tahoe, an early version of TCP. A newer version of TCP, TCP Reno, makes a distinction between a loss event caused by a timeout and a loss event caused by receiving three duplicate acknowledgments [103]. In TCP Reno, when three duplicate acknowledgments are received, instead of reducing the *cwnd* to 1 MSS and entering slow start, the algorithm simply sets the *cwnd* to *ssthresh* and begins the congestion avoidance process. This avoidance of slow start when three duplicate acknowledgments are received is know as "fast recovery." Other versions of TCP also exists; see [57] for a discussion of more recent developments in TCP congestion control.

APPENDIX B
Traceroute of Internet DOORS Case

The following hop list was obtained for the DOORS Internet Case by running traceroute from the client machine (in the Computer Science lab) to the IP address of the Linksys Etherfast Cable/DSL Router (in the home network). Machine names have been removed but each machine is identified in the trace by an IP address.

```
 1  (128.213.16.1)       0.370 ms   0.163 ms   0.141 ms
 2  (128.213.8.254)      0.675 ms   0.633 ms   0.629 ms
 3  (128.213.8.14)       0.387 ms   0.361 ms   0.382 ms
 4  (128.113.113.254)    1.069 ms   1.018 ms   0.870 ms
 5  (128.113.39.251)     3.863 ms   4.241 ms   3.303 ms
 6  (169.130.253.65)     3.579 ms   3.227 ms   3.307 ms
 7  (169.130.3.29)       6.309 ms   7.233 ms   7.700 ms
 8  (204.148.99.13)     12.664 ms  10.535 ms  11.400 ms
 9  (204.148.101.221)   13.037 ms  12.862 ms  13.890 ms
10  (64.236.7.173)      16.564 ms  16.208 ms  16.860 ms
11  (64.236.4.2)        18.295 ms  18.786 ms  20.359 ms
12  (64.236.4.242)      18.657 ms  19.991 ms  20.562 ms
13  (24.29.33.27)       20.130 ms  22.482 ms  23.276 ms
14  (24.29.32.122)      26.288 ms  24.603 ms  22.217 ms
15  (24.161.46.213)     32.393 ms  32.097 ms  29.595 ms
```

APPENDIX C
Separable Flows

C.1 Introduction

As companies increase their connectivity and network infrastructure, the number of network applications for both intra-company organization and inter-company commerce continue to grow. This trend increases the need for scalability analysis of these applications in terms of the size of the application and the network involved. A great deal of research has been done on the scalability of parallel algorithms and applications as a function of the parallel architecture size (e.g., the number of processors used, etc.) [173, 101, 70, 188]. However, scalability analysis for network applications creates a different set of challenges and must therefore rely on new metrics and techniques.

Simulation-based scalability analysis provides the middle ground between fast and inexpensive analytical methods that may be imprecise and real-life experiments that are very precise but expensive and time-consuming to perform. Simulation-based scalability analysis provides the following benefits.

- **It creates a consistent basis for comparisons.** Comparative scalability analysis often requires comparing the performance of two different ways of performing a similar action. A real-time, laboratory-based experiment could be of used for this purpose, but recreating the same execution environment for two executions is difficult to achieve. Experiments in isolated laboratory will be limited in size, and experiments on large production network will be influenced by extraneous flows that are not controllable.

- **It can identify favorable as well as undesirable network/traffic configurations.** A study of what network and traffic conditions work best for the particular application provides insight into the strengths of the application as well as any optimizations that may be beneficial. The bounds of the application can also be investigated. Determining when the application breaks down

or is no longer beneficial can be an important process when designing fault tolerant systems.

- **It can quantify benefits and costs of different configurations.** The simulation parameters can easily be varied to represent different network topologies and traffic conditions. In most simulators, many metrics and application characteristics can be easily recorded, providing the basis for a multi-dimensional scalability or optimization analysis.

Others have used simulation for scalability analysis of network applications [62, 61]. However, these works involved elaborate customizing of a general simulator to model the complex behavior of network applications. Authors of [47, 46] quantitatively model the interactions of parallel and sequential jobs on a network of workstations. Although this work is for an optimization of a different problem, it is important to our work for further understanding and quantifying the interactions of different flows on each other.

Simulating an application's flow in a network simulator can be very difficult because of the various points of application synchronization. To avoid the hardships of changing a significant amount of the simulator code, we propose a method of finding appropriate segmentation points whereby we can divide the flow of the application into smaller, easily simulated flows. For example, a simple two-level connection architecture may involve a client connecting to a server. Before the connection is closed, the server connects to a second server to gather information. In this case, the application synchronization points can help to serve as the flow endpoints for each simulation. Many applications execute a sequence of stages. Flows in each stage happen at non-intersecting times. Often, the flows themselves happen at separate, non-intersecting physical locations in the network. Even if some flows are active at the same time and possibly at the same place, they can still be modeled separately if their interactions are negligible. Flows that can be separated in this fashion are referred to as *separable*.

In this appendix, we will first discuss other uses of separability in networking. We will then review a basic example of separable flows, followed by the our methodology. Lastly, we present a formal analysis of separable flows with our conclusions.

C.2 Separability

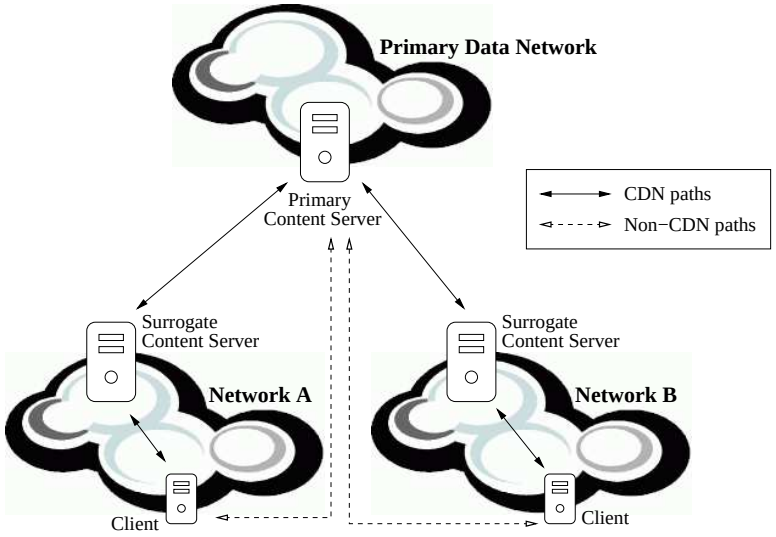
Separability in networking is not an entirely new concept. For years researchers have had to prove some form of queuing separability to solve queuing circuits [72]. This form of separability essentially meant that it must be possible to evaluate the performance measures of the complete circuit of queuing centers as though each of the centers were evaluated in isolation. The performance of the circuit as a whole can then be evaluated by a statistical or mathematical combination of these separate solutions [16]. Kleinrock describes the concept of separable performance measures as those that may be expressed simply as a sum of terms, each of which depends only on the flow in a single channel [98]. An extensive discussion of separable queuing network models, including their requirements, limitations, and extensions, can be found in [105].

While the purpose of separability in queuing models is different than our motivation for separable flows in network applications, they share many similarities. Separable flows enable the modeler to segment the different flows of an application into separate, easily simulated flows. Much like the separable queuing networks, the results of the separate simulations can then be statistically or mathematically combined to compute overall performance metrics for the entire application. The type of combination needed depends on the type of metrics involved. For example, if flow source-destination delay is of interest, it could be obtained by summing of all separately simulated flow delays.

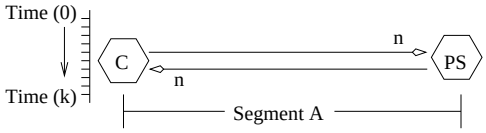
C.3 Example

Figure C.1(a) shows the architecture of a simple Content Distribution Network (CDN) provider. To quantify the benefit of CDN provider, the delay of a flow using CDN surrogate servers must be compared to the delay of a flow using Internet paths. These two flows are presented in Figures C.1(b and c).

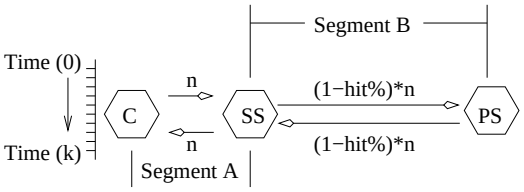
The Internet path (Figure C.1(b)) represents a simple request/reply communication between a client and a server, so it can be easily simulated directly (without segmentation). If there are n requests in one direction, there would be n responses in the opposite direction. The CDN path (Figure C.1(c)), involving a client, server,



(a) Simple Content Distribution Network(CDN) architecture



(b) Internet path application flow



(c) Surrogate Server path (CDN) application flow

Figure C.1: Architecture and flow diagrams for CDN application.

and a surrogate server, is a bit more complicated. In the CDN path, the client would issue n requests to a surrogate server, out of which $(hit\_ratio * n)$ would be successfully replied by the surrogate with no need to go to the primary server. However, the surrogate server would also need to pause its connection with the client while seeking the original information from the primary server $(1 - hit\_ratio) * n$ times. This type of behavior may not be easily modeled in most generic simulators. Therefore we need to segment the flows at the synchronization point (the surrogate server) for separable simulations. If the total delay is needed, it can be calculated as follows.

$$Delay_{total} = Delay_A + (1 - hit\_ratio) * Delay_B \tag{C.1}$$

C.4 Methodology

Let T denote the vector describing the network topology and its link capacities (hardware). Let $S_A = < S_1^A, S_2^A, ..., S_k^A >$ denote activities of the sources of the application traffic. Finally, let $S_B = < S_1^B, S_2^B, ..., S_m^B >$ denote activities of the background traffic. The simulation is a function σ, that for a given vector $< T, S_A, S_B >$, uniquely defines the application and background flows denoted by $F_A =< f_1^A, f_2^A, ..., f_k^A >$ and $F_B =< f_1^B, f_2^B, ..., f_m^B >$ (the relationship between σ, F_A, and F_B is shown in Equation C.2).

$$< F_A, F_B >= \sigma(T, S_A, S_B) \tag{C.2}$$

A simulation with a separated flow F_i is simply run with all the application sources, except the i^{th} one, turned off. Thus, source activities for such simulations are defined by the vector $S_i =< \phi, ..., S_i^A, ..., \phi >$ with a result of $< F_i, F_B^i >= \sigma(T, S_i, S_B)$.

Let M(F) be the metric of measurements obtained for flow F. For the given background flows B the relative error of separation of flow F_i from F_A is

$$\epsilon = \max_{1 \le i \le k} \frac{|M(F_A) - M(F_i)|}{M(F_A)}$$

In general, as discussed earlier, the relative error is small if the separated flows do not interact with each other, i.e., if they do not share network nodes at all or use them at different time intervals. If the metric of interest is a flow delay, the relative error is also small (and the application flows are separable) in the following two cases of sharing.

- **Case (1) Utilization on network nodes shared by the separated flows is low.** In this case, the networking nodes which the flows share are underutilized and therefore the interaction of flows on these nodes does not significantly affect either flow. Indeed, for an M/M/1 system, such as a router queue with Poisson inflow, the delay for a server with the processing rate μ and the in-flow λ is $1/(\mu - \lambda) = 1/\mu(1 + u + u^2 \ldots)$, where u is the server utilization. For small utilization, the relative error of approximating application delay by a

separated flow delay is at most u, hence, small.

- **Case (2) Utilization on network nodes shared by the separated flows is medium, but the application flow rate is small compared to the background traffic flow rate.** In this case, a network node that the flows share can be well utilized, but the impact of flows F on the queue size of this node is negligible, so the queue contains mainly background flow packets. For an M/M/1 system, repeating the above analysis, we can establish that the relative error of approximation is about $\lambda_i/(\mu + \lambda)$, where λ_i is the rate of one of the separated flows and λ is the rate of the background flow. Since $\lambda_i < \lambda < \mu$, this relative error is small. However, it is important that the server utilization is not high, otherwise even a small change in the inflow will create a large increase in the total delay. This effect is illustrated later in the appendix.

C.4.1 Quantifying Flow Interactions

Flows, for the purpose of segmentation capability, can be described as sequences of annotated packets. Figure C.2 shows the contents of a flow as a list of q annotated packets and their locations at m discrete times.

$$F_A = \begin{bmatrix} P_0((t_{0,0}, c_{0,0}), (t_{0,1}, c_{0,1}), \cdots, (t_{0,m}, c_{0,m})) \\ P_1((t_{1,0}, c_{1,0}), (t_{1,1}, c_{1,1}), \cdots, (t_{1,m}, c_{1,m})) \\ \vdots \\ P_q((t_{q,0}, c_{q,0}), (t_{q,1}, c_{q,1}), \cdots, (t_{q,m}, c_{q,m})) \end{bmatrix}$$

Figure C.2: Annotated packets flow description

In Figure (C.2), $P_a(t, c)$ describes the location "c" of packet "a" at discrete time "t" and each location is in the set of all network components (including hosts, routers, and links) that a packet may reside at any point in time. Therefore, the interaction of one flow on another can be found by performing a conditional "join" ($\bowtie$) of the two flows, revealing the packets which are at the same networking component within the same or adjacent time periods (Figure C.3).

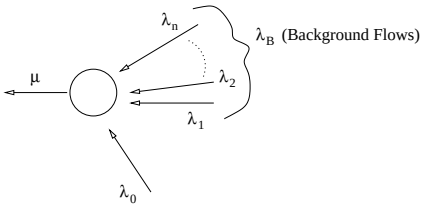
$$\begin{bmatrix} P_0((t_{0,0}, c_{0,0}), \cdots, (t_{0,m}, c_{0,m})) \\ P_1((t_{1,0}, c_{1,0}), \cdots, (t_{1,m}, c_{1,m})) \\ \vdots \\ P_q((t_{q,0}, c_{q,0}) \cdots, (t_{q,m}, c_{q,m})) \end{bmatrix} \bowtie \begin{bmatrix} P'_0((t'_{0,0}, c'_{0,0}), \cdots, (t'_{0,m}, c'_{0,m})) \\ P'_1((t'_{1,0}, c'_{1,0}), \cdots, (t'_{1,m}, c'_{1,m})) \\ \vdots \\ P'_q((t'_{q,0}, c'_{q,0}), \cdots, (t'_{q,m}, c'_{q,m})) \end{bmatrix}$$

Figure C.3: Description of the join operation between two annotated packet flows

The interaction defined on the packet level is negligible unless packets from one flow enter a networking component within $(q_c * x_c)$ where q_c is the size of the queue at component c, and x_c is the average service time at component c. In essence, the interaction between the two is negligible, if the device has had enough time to totally empty its queue. Formally, this interaction can be defined with respect to annotated packet flows in Equation (C.3).

$$P_x(t_{a,i}, comp_{x,c}), P'_y(t_{b,j}, comp_{y,c}) \; and \; |t_{a,i} - t_{b,j}| < q_c * t_{p,c} \tag{C.3}$$

C.5 Formal Analysis of Separability for Flow Delay Metric



(a) *M/M/1* Queuing illustration of separable flow

λ	total flows to the router (including application whose flow is to be segmented). $\lambda = \lambda_0 + \lambda_B$, where λ_B is all flows but separation candidate λ_0.
μ	service rate of the server.
$\Delta\lambda$	half the range of the expected application flow variability.

(b) Table of symbols

Figure C.4: Queuing illustration and symbol table

We assume that utilization, $u = \frac{\lambda}{\mu}$, is modest (less than 50%) and utilization from the application alone is small (10% or less). We also assume that the behavior of separable candidate flow λ_0 is bounded by $\lambda_{0,Min} = \lambda_0 - \Delta\lambda$ and $\lambda_{0,Max} = \lambda_0 + \Delta\lambda$ in

such a way that $\lambda_{0,Max} = \lambda_{0,Min} + 2 * \Delta\lambda$, where $\lambda_{0,Max}$, and $\lambda_{0,Min}$ are the maximum and minimum flows respectfully. We assume a *M/M/1* system, so the delay t in the system (including waiting in the queue and the service) is $t = \frac{1}{\mu - \lambda}$.

The change in this delay observed under maximum and minimum flows from the application is described in (C.4).

$$\begin{aligned} \Delta t &= \frac{1}{\mu - (\lambda + \Delta\lambda)} - \frac{1}{\mu - (\lambda - \Delta\lambda)} \\ &= \frac{(\mu - \lambda + \Delta\lambda) - (\mu - \lambda - \Delta\lambda)}{(\mu - \lambda + \Delta\lambda)(\mu - \lambda - \Delta\lambda)} \\ &= \frac{2 * \Delta\lambda}{(\mu - \lambda)^2 - \Delta\lambda^2} \end{aligned} \tag{C.4}$$

According to our assumptions that overall utilization is below 50% and the utilization caused by the application flow is small (below 10%), the overall utilization and application utilization can expressed as the function of the respective flow λ's:

$$u_{overall} = \frac{\lambda_{overall}}{\mu} < 0.5 \quad \Rightarrow \quad \lambda_{overall} \leq 0.5 * \mu \tag{C.5}$$

$$u_{Flow0} = \frac{\lambda_{Flow0}}{\mu} < 0.1 \quad \Rightarrow \quad \lambda_{Flow0} \leq 0.1 * \mu \tag{C.6}$$

Therefore

$$\begin{aligned} & & 0.1\mu \geq \lambda_{0,Max} = \lambda_{0,Min} + 2 * \Delta\lambda \\ (\mu - \lambda)^2 = (\mu - 0.5\mu)^2 & & 0.05\mu \geq \Delta\lambda \\ (\mu - \lambda)^2 > 0.25\mu^2 & & \Delta\lambda^2 \leq 0.0025\mu^2 \end{aligned} \tag{C.7}$$

But $(\mu - \lambda)^2 >> \Delta\lambda^2$, so (C.4) reduces to (C.8).

$$\Delta t = \frac{2 * \Delta\lambda}{(\mu - \lambda)^2 - \Delta\lambda^2} \approx \frac{2 * \Delta\lambda}{(\mu - \lambda)^2} \tag{C.8}$$

To study the effects of the flow in question upon other flows, we look for values

of $\frac{\Delta t_{Flow0}}{t}$ that are small:

$$\frac{\Delta t_{Flow0}}{t} = \frac{\frac{2*\Delta\lambda}{(\mu-\lambda)^2}}{\frac{1}{\mu-\lambda}} = \frac{2*\Delta\lambda}{\mu-\lambda} = \frac{2*\Delta\lambda}{\frac{\lambda}{u}-\lambda} = \frac{\lambda_{0,Max}}{\frac{\lambda}{u}-\lambda} = \frac{\lambda_{0,Max}*u}{\lambda*(1-u)} = \frac{\lambda_{0,Max}}{\lambda} * \frac{u}{(1-u)} \quad \text{(C.9)}$$

The $\frac{u}{(1-u)}$ portion of Equation (C.9) is small for $u << 1$, so if utilization is low, it does not matter what the value of $\frac{\lambda_{0,Max}}{\lambda}$ may be. On the other hand when utilization is getting close to modest levels, more than 10% but less than 50%, then $\frac{u}{(1-u)}$ is close to 1, so the fraction $\frac{\lambda_{0,Max}}{\lambda}$ must be small. Assuming that we want to have our results within v=10% of their real value, we can write:

$$v \geq \frac{\Delta t}{t} = \frac{u}{1-u} * \frac{\lambda_{0,Max}}{\lambda}$$

We can plot $\frac{\lambda_{0,Max}}{\lambda}$ as a function of the utilization (u) and the strictness of segmentation approximation v.

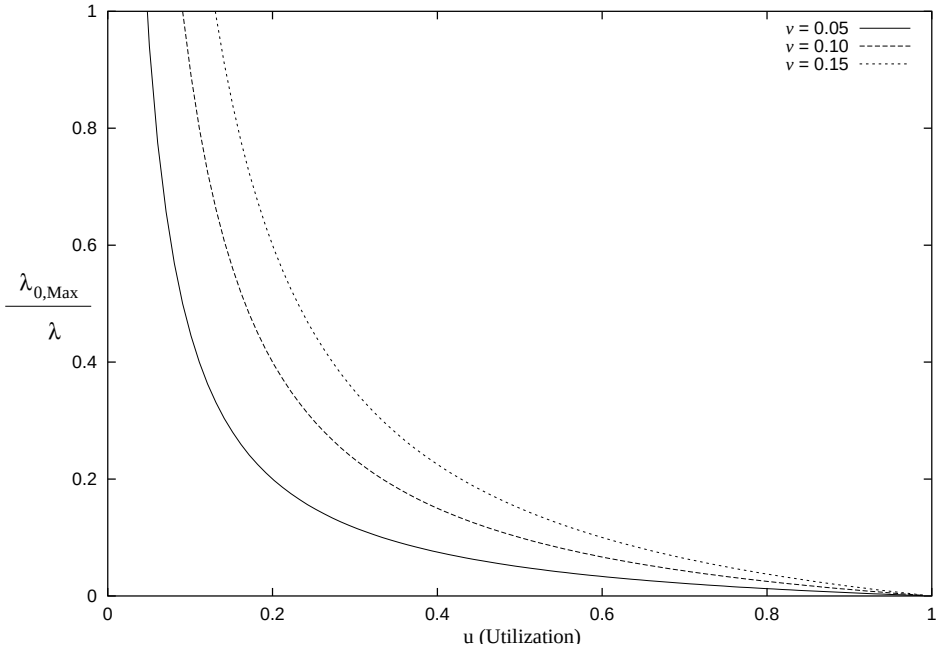


Figure C.5: Plot of $\frac{\Delta t_{Flow0}}{t}$: Points below the curve indicate a separable flow.

The curves are a family of hyperbole (cf. Figure C.5). As seen in Figure C.5, when the utilization is low, the candidate flow is separable. When the utilization is

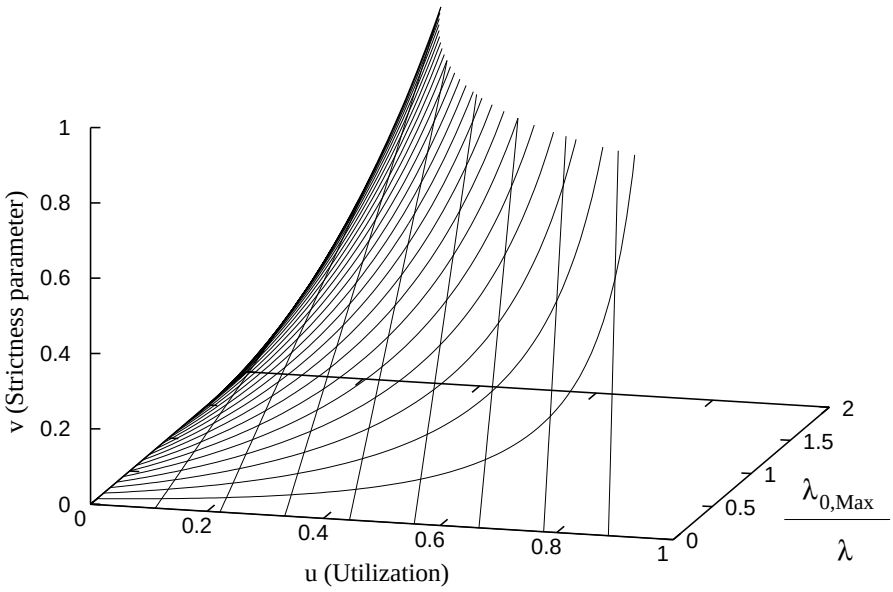


Figure C.6: 3-Dimensional plot of $\frac{\Delta t_{Flow0}}{t} \leq v$: Points above the curved plane are strictness parameters of segmentation.

high, the ratio $\frac{\lambda_{0,Max}}{\lambda}$ must be low in order for the candidate flow to be separable. All points below the curve are separable. Figure C.6 displays a three-dimensional graph in which v changes along a vertical axis. The points on the surface and above shows the values of the strictness parameter v for which the candidate flow is separable with the given values of $\frac{\lambda_{0,Max}}{\lambda}$ and u. When the utilization is minimal, even for the small values of v, the flow is separable. Interestingly, even at strictness of 100% (i.e., when the candidate flow is considered separable even if it changes the total traffic by 100%) no combinations of high utilization and high values of $\frac{\lambda_{0,Max}}{\lambda}$ can make the candidate flow separable.

C.6 Summary

In this appendix, we have explained the benefits of simulation-based scalability analysis of network applications. We have shown how very complex network

applications can be segmented into smaller flows for separate simulation of each flow. We have also shown through formal analysis, that in cases of certain hardware utilization levels and flow influence values, applications can be proven separable, without other forms of information.

APPENDIX D
Frequency-Based Clustering for Self-Organizing Maps

Many complex problems are difficult to organize and process due to their large number of changing parameters. Learning strategies such as neural networks have shown to be effective for modeling complex problems, as they are created to mimic the way the human brain handles numerous inputs. A special type of neural network, the Self-Organizing Map (SOM) is especially effective when unsupervised learning is required. In this appendix, we will explain the algorithms used in forming SOMs, and a new automated clustering algorithm we have created which uses the SOM as its base.

Self-organizing maps are often used for novelty detection [183], automated clustering [125, 172], and visual organization [99]. They are particularly useful for highly multidimensional data because the data vectors are stored in neurons that are mapped to a one dimensional line, or two dimensional Cartesian plane (higher dimensions are possible, but usually not used). This is also why the SOM is such a powerful visual clustering tool. Because the neurons are usually arranged in less than three dimensions, they are easy to visualize.

D.1 Description of Self-Organizing Maps

For the purpose of this appendix we will only focus on self-organizing maps topologically arranged in two dimensional space. Such a SOM is composed of a grid of neurons (often arranged into a rectangular). Each neuron, $m_{x,y}$, has a vector of weights associated with it. The weights of the neurons are to represent the input values; therefore, they must be the same in number and magnitude. Many times implementors will give the neurons initial random values. The SOM must be trained to accurately represent the input vectors topologically.

D.2 Training the SOM

The theory of self-organizing maps is beyond the scope of this appendix; however, we will discuss some of the SOM fundamentals as they pertain to our automated clustering implementation. A theoretical foundation of SOMs can be found in [99].

The SOM is trained by presenting the certain number k of input vectors to the map and updating the closest neuron and the neuron's neighbors to make their weights closer to the values in the input vector. The training algorithm selects input vectors $x_j = [\xi_{j1}, \xi_{j2}, \cdots, \xi_{jn}]$, and computes the Best Matching Unit (BMU) for each input vector in the SOM. The BMU is determined by simply applying the Euclidean distance formula to determine which neuron's weight vector $m_i = [\mu_{i1}, \mu_{i2}, \cdots, \mu_{in}]$ appears to be closest to the input vector j (See Equation D.1).

$$BMU = i \; that \; minimizes \left(\sum_{x=1}^{n} (\xi_{jx} - \mu_{ix})^2 \right) \tag{D.1}$$

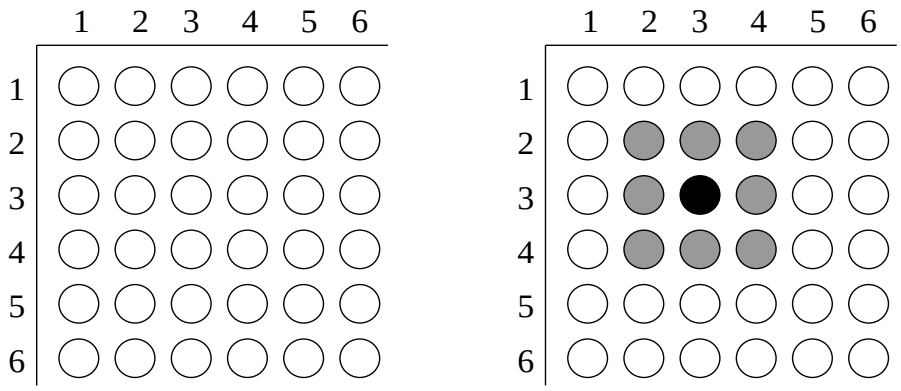
Once the BMU has been determined, it along with its neighbors must be updated. The update is performed according to Equation D.2.

$$m_i(t+1) = m_i(t) + \alpha(t)[x(t) - m_i(t)] \quad for \; each \; i \in N(t) \tag{D.2}$$

where:

- $\alpha(t)$ is a linearly non-increasing learning function $\alpha(t) \in [0, 1]$
- $N(t)$ is a linearly non-increasing neighborhood function
  (in our example, $N(t) \in [0, 8]$)

An illustration of the SOM and a computed BMU with surrounding neighbors can be found in Figures D.1(a) and D.1(b). In these figures the SOM has a 6x6 neuron grid (Figure D.1(a)) and the BMU was calculated to be the neuron at position (3,3). In this case, the neighborhood function $N(t)$ returned eight, so eight of the surrounding nodes were selected for updating (Figure D.1(b)). The process of picking a new input vector, determining the BMU, and updating the SOM neurons based on Equation D.2 continues until the process causes little change in the weights



(a) Neuron Grid

(b) Neighborhood of the BMU

Figure D.1: Illustration of the SOM and the updating process

of the neurons. This results in a topological ordering of the SOM neurons, with weight vectors representing similar input vectors appearing close to each other.

D.3 Clustering with the SOM

D.3.1 Visual Clustering

Many algorithms exist for visually representing the two-dimensional neuron grid in the SOM. Most focus on a function of the Euclidean difference from one neuron's weight vector to another. The larger the difference, the darker they make the neuron [99]. Some also choose to display the SOM matrix using a simple three dimensional histogram. The advantage is that the SOM provides a way to graphically display data with many dimensions on a two dimensional plane.

D.3.2 Automated Clustering

To automate the clustering process, we use a frequency-based approach to develop cluster centers called centroids. We compute a frequency value β to count how many times a particular neuron and members of its neighborhood were chosen as the BMU during training. The frequency value is computed so that the BMU instances of neighboring nodes contribute less as their distance from the prospective

centroid increases. After training, the neurons with the highest frequency value (β), are selected to be centroids. Our calculation of β is shown in Equation D.3.

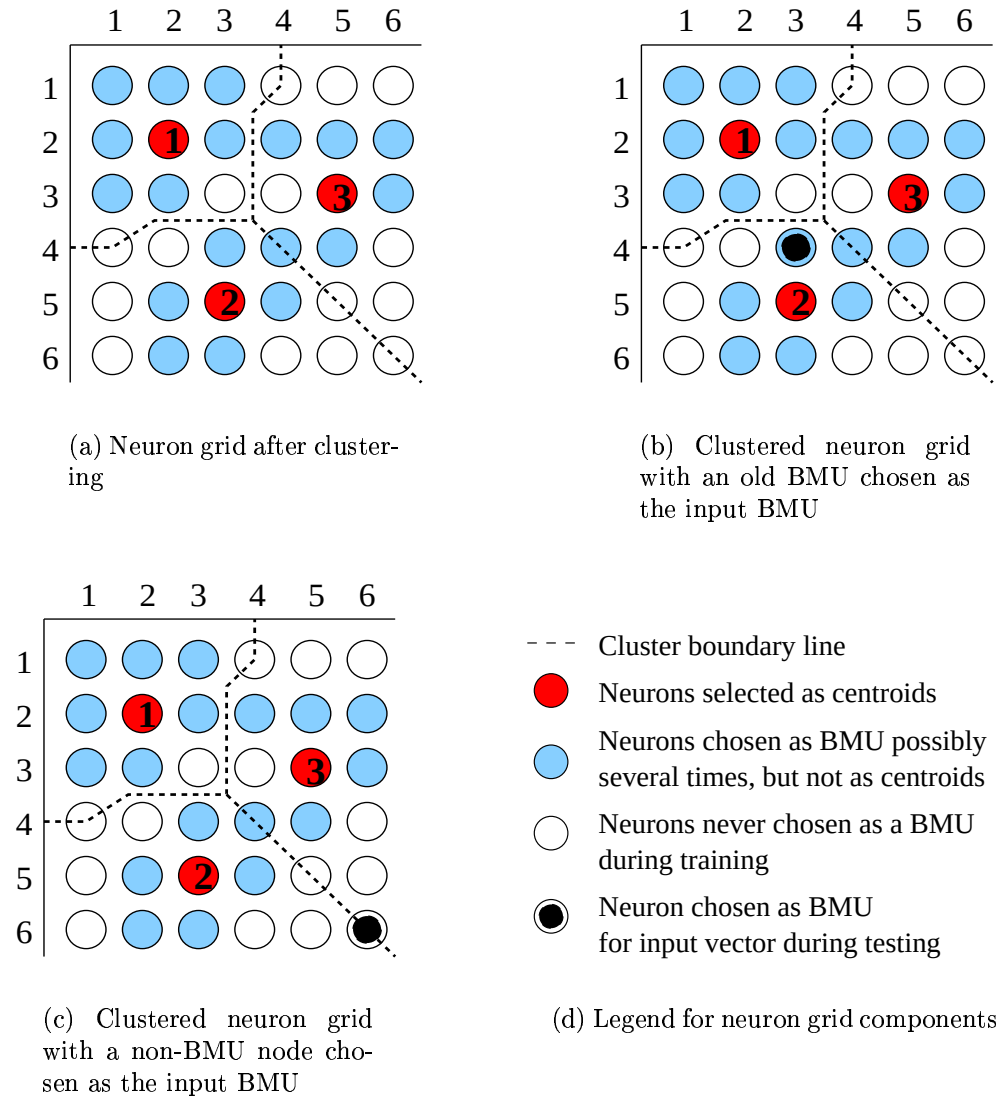
$$\beta_{i,j} = \sum_{x=-reach}^{reach} \sum_{y=-reach}^{reach} \frac{freq_{i+x,j+y}}{1+\sqrt{x^2+y^2}} \tag{D.3}$$

for all valid neurons, where:

- $freq_{i,j}$ = number of times the neuron at position (i,j) was the BMU
- $reach$ = the spacing we enforce between cluster centroids

If our algorithm is given the number of clusters to generate (z), it simply chooses the z neurons with the highest β and not adjacent to current centroids to be centroids of the z clusters. In this context, the neuron at position (x_1,y_1) is adjacent to the neuron at position (x_2,y_2) if ($x_2 - reach \leq x_1 \leq x_2 + reach$) and ($y_2 - reach \leq y_1 \leq y_2 + reach$). Once again, $reach$ is an integer representing the spacing we enforce between cluster centroids. Once a neuron is chosen to be a centroid, the β value for all other non-centroid nodes are recalculated to avoid the influence of the new centroid on the formation of another cluster. The new β value is computed according to Equation D.3; however, while performing the calculations, any centroid node frequency values are removed from the computation. We have additional algorithms that we use if the number of desired clusters is unknown.

After the learning phase, to determine what cluster a particular input belongs to, we simply compute the BMU for the input and select the cluster which has the closest centroid (according to Euclidean distance). Figure D.2 illustrates this process. Once the SOM has been trained and the centroids have been selected, the SOM matrix can be logically divided into clusters based on the grid distance of each node from the centroids. An illustration of this division is shown by the dashed line in Figure D.2(a). Notice that in this diagram, some nodes are equidistant from two centroids. In this case, the neuron has a membership in both clusters and further processing is necessary to determine which group will be chosen. Figure D.2(b) shows an example of the BMU calculated from an input vector presented to the SOM for classification. In this case, the input would be clustered into cluster 2 because it is closest to cluster 2's centroid.



(a) Neuron grid after clustering

(b) Clustered neuron grid with an old BMU chosen as the input BMU

(c) Clustered neuron grid with a non-BMU node chosen as the input BMU

(d) Legend for neuron grid components

Figure D.2: Illustration of the SOM clustering process

This clustering method is sensitive to the learning process of the SOM. An ambiguous situation may occur when clustering, if an input vector is received that maps to a SOM neuron whose weights have never been adjusted in the learning process. This special case would mean that the neuron would not have been chosen as the BMU or been a neighbor affected by the updating of a close BMU. An example of this special case is shown in Figure D.2(c). This case is ambiguous because there is no reason to believe the chosen neuron, still containing its initial values, should

belong in any group. In Figure D.2(c), the neuron selected to be the closest match to the input vector has actually never been a training BMU. It can also be noted that it could not have been affected in the training cycle because no surrounding neighbors were BMU's either. This is the only node in the neuron matrix where this could be the case. An input vector matching to this node would be an ambiguous and non-classifiable case. One way of resolving this issue is to use better initialization techniques to ensure a more uniform BMU distribution over the entire map.

Future work is necessary to further fine-tune this technique, if it were to become more popular. However, our current prototype was sufficient enough to cluster data for our needs.

APPENDIX E
Glossary of Commonly Used Acronyms

This Appendix is a glossary of acronyms frequently used in the text of the thesis. Some of these acronyms were created by our group, and will be identified with a special superscript denoting which work the acronym is a part of. A quick reference for the superscripts used is found in Table E.1.

Superscript	Description
DOORS	Section 2.2 DOORS: Distributing Network Applications
NNCONG	Section 5.2 Neural Network Congestion Arbitration and Source Prediction
NNID	Section 3.3 Artificial Neural Networks for Denial of Service Detection
GENRE	Section 7.2.1.3 GENRE
PSFA	Section 4.2 Probabilistic State Finite Automata Host-based Intrusion Detection
TDFA	Section 3.2 Time Dependent Finite Automata for Denial of Service Detection

Table E.1: A description of superscripts used in the glossary

Glossary

Acronym		Meaning
ABR	-	Available Bit-Rate
ACK	-	Acknowledgment
AQM	-	Active Queue Management
ASCII	-	American Standard Code for Information Interchange
ATM	-	Asynchronous Transfer Mode
BMU	-	Best Matching Unit
BSD	-	Berkeley Software Distribution
BSM	-	Basic Security Module

CAC	-	Connection Admission Control
CBR	-	Constant Bit Rate
CIM	-	Common Information Model
CMIP	-	Common Management Information Protocol
COPS	-	Common Open Policy Service
CPU	-	Central Processing Unit
CFT^{NNID}	-	Command Frequency Table
CSMA/CD	-	Carrier Sense Multiple Access with Collision Detection
CTIT	-	Centre for Telematics and Information Technology
DARPA	-	Defense Advanced Research Projects Agency
DCN	-	Data Communication Network
DDOS	-	Distributed Denial of Service network attack
DFA	-	Deterministic Finite Automaton
DFU^{TDFA}	-	Data Filtration Unit
DMTF	-	Distributed Management Task Force
DOORS	-	Distributed Online Object Repositories
DoS	-	Denial of Service network attack
ECN	-	Early Congestion Notification
ETG^{TDFA}	-	Event Token Generator
FRED	-	Flow-based Random Early Detection
ICMP	-	Internet Control Message Protocol
IDS	-	Intrusion Detection System
IEEE 802.3	-	Institute of Electrical and Electronics Engineers Ethernet Standard
IP	-	Internet Protocol
ISP	-	Internet Service Provider
kNN	-	k-Nearest Neighbor
LAN	-	Local Area Network
MF	-	More Fragments (IP Header Field)

MIN	-	Multistage Interconnection Network
MLP	-	Multi-layer Perceptron
MSS	-	Maximum Segment Size
NFS	-	Network File System
OSI	-	Open Systems Interconnection
PSFA	-	Probabilistic State Finite Automata
QoS	-	Quality of Service
R2L	-	Remote to Local attack
RED	-	Random Early Detection or Random Early Drop
RIP	-	Routing Information Protocol
RMON	-	Remote Monitoring
ROC	-	Relative Operating Characteristic
SEA	-	Schonlau et al (used to refer to their masquerade detection dataset)
SLA	-	Service Level Agreement
SNA	-	Systems Network Architecture
SNMP	-	Simple Network Management Protocol
SOM	-	Self-Organizing Map
SSO^{TDFA}	-	Site Security Officer
SYN	-	Synchronization Packet
TCP	-	Transmission Control Protocol
TDFA	-	Time Dependent Finite Automata
TMN	-	Telecommunications Management Network
TTU^{TDFA}	-	TDFA Transversal Unit
U2R	-	User to Root attack
UDP	-	User Datagram Protocol
VPN	-	Virtual Private Network
WBEM	-	Web-Based Enterprise Management

LITERATURE CITED

[1] Robert K. Ackerman. Government enlists industry for information security. In *Signal*, volume 56, pages 17–20. Armed Forces Communications and Electronics Association, 4400 Fair Lakes Court, Fairfax, VA 22033, August 2002.

[2] Jong Suk Ahn, Peter B. Danzig, Zhen Liu, and Limin Yan. Evaluation of TCP vegas: Emulation and experiment. In *Proceedings of ACM SIGCOMM '95*, pages 185–205, Boston, MA, August 1995. http://www.acm.org/sigcomm/sigcomm95/papers/ahn.html.

[3] D. Scott Alexander. The switchware active network architecture. *IEEE Network Special Issue on Active and Controllable Networks*, 12(3):29–36, July 1998.

[4] M. Allman, V. Paxson, and W. Stevens. TCP congestion control. Request for Comments: 2581, April 1999.

[5] Mark Allman, Chris Hayes, and Shawn Ostermann. An evaluation of TCP with larger initial windows. Technical Newsletter: Computer Communication Review, July 1998.

[6] Edward Amoroso. *Intrusion Detection: An Introduction to Surveillance, Correlation, Traps, Trace Back, and Response.* Intrusion.Net Books, Sparta, New Jersey, 1 edition, June 1999.

[7] Mandy Andress. Denial of service: Fighting back. In *NetworkWorld.* Network World Inc., 118 Turnpike Road, Southborough, MA 01772, September 2002.

[8] Alex Aussem, Antoine Mahul, and Raymond Marie. Queueing network modelling with distributed neural networks for service quality estimation in b-isdn networks. In *Proceedings of IEEE-INNS-ENNS International Joint Conference on Neural Networks*, pages 392–397, Como, Italy, 2000.

[9] S. Axelsson. Research in intrusion-detection systems: A survey. Technical Report 98–17, Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden, December 1998.

[10] B.R. Badrinath and Pradeep Sudame. Transformer tunnels: A framework for providing route-specific adaptations. In *The USENIX Annual Technical Conference*, New Orleans, Louisiana, June 1998. USENIX.

[11] Prasad Bagel, Shivkumar Kalyanaraman, Shrikrishna Karandikar, and Bob Packer. TCP rate control. Technical Newsletter: Computer Communication Review, January 2000.

[12] Sandeep Bajaj, Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, Padma Haldar, Mark Handley, Ahmed Helmy, John Heidemann, Polly Huang, Satish Kumar, Steven McCanne, Reza Rejaie, Puneet Sharma, Kannan Varadhan, Ya Xu, Haobo Yu, and Daniel Zappala. Improving simulation for network research. Technical Report 99-702, University of Southern California, Los Angeles, CA, March Improving Simulation for Network Research.

[13] Sean Baker. *CORBA Distributed Objects using Orbix.* Addison Wesley, 1515 Broadway, 17th Floor, New York, NY 10036-5701, 1997.

[14] M. Baldi, S. Gai, and G. P. Picco. Exploiting code mobility in decentralized and flexible network management. In *Mobile Agents*, pages 13–26, Berlin, Germany, April 7-8 1997. First International Workshop, MA, Springer Verlag.

[15] André Mello Barotto, Andriano de Souza, and Carlos Becker Westphall. Distributed network management using SNMP, Java, WWW and CORBA. *Journal of Network and Systems Mnagement*, 8(4):483–497, 2000.

[16] Forest Baskett, K. Mani Chandy, Richard R. Muntz, and Fernando G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the Associaton for Computing Machinery*, 22(2):248–260, April 1975.

[17] M. A. Bauer, R. B. Bunt, A. El Rayess, P. J. Finnigan, T. Kunz, H. L. Lutfiyya, A. D. Marshall, P. Martin, G. M. Oster, W. Powley, J. Rolia, D. Taylor, and M. Woodside. Services supporting management of distributed applications and systems. *IBM Systems Journal*, 36(4):508–526, 1997.

[18] Stephen Billips. Final report of the mathematics clinic: Automatic calibration of water distribution network models. http://www-math.cudenver.edu/clinic/reports/Spring1998.html, 1998.

[19] A. Bivens, P. Fry, L. Gao, M.F. Hulber, B. Szymanski, and Q. Zhang. Distributed object-oriented repositories for network management. In *Proc. 13th Int. Conference on Software Engineering*, pages CS169–174, Las Vegas, NV, August 1999.

[20] J. Alan Bivens, Li Gao, Mark Hulber, and Boleslaw Szymanski. Agent-based network monitoring. In *Agent based High Performance Computing*, Seattle, Washington, May 1999. Autonomous Agents: Sponsered by ACM SigART.

[21] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on queue management and congestion avoidance in the internet. RFC 2309, April 1998. Category: Informational.

[22] L. Brakmo and L. Peterson. TCP vegas: End to end congestion avoidance on a global internet. *IEEE Journal of Selected Areas in Communications*, 13(8):1465–1480, October 1995. ftp://ftp.cs.arizona.edu/xkernel/Papers/jsac.ps.

[23] K. L. Calvert. Architectural framework for active networks. RFC Draft, July 1999.

[24] Peter K. Campbell, Alan Christiansen, Michael Dale, Herman L. Ferr'a, Adam Kowalczyk, and Jacek Szymanski. Experiments with simple neural networks for real-time control. *IEEE Journal on Selected Areas in Communications*, 15(2):165–178, February 1997.

[25] James Cannady. Artificial neural networks for misuse detection. In *1998 National Information Systems Security Conference*, pages 443–456, Arlington, VA, October 1998. (NISSC'98).

[26] J.D. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Structure of management information for version 2 of the simple network management protocol. RFC 1902, January 1996.

[27] E. Casilari, A. Alfaro, A.Reyes, A. Dmaz-Estrella, and F. Sandoval. Neural modelling of ethernet traffic over atm networks. In *the Proceedings of the fourth International Conference on Engineering Applications of Neural Networks (EANN'98)*, pages 304–307, Gibraltar, June 1998.

[28] CCITT. Specification of abstract syntax notation one (asn.1). CCITT Recommendation X.208:, 1988.

[29] CCITT. Specification of basic encoding rules for abstract syntax notation one (asn.1). Recommendation X.209, 1988.

[30] CCITT. Recommendation m.3010, principles for a telecommunications management network. CCITT: Recommendation M.3010, 1996. Geneva.

[31] S. Chaudhuri and V. Narasayya. Autoadmin "what-if" index analysis utility. In *Proceedings of ACM SIGMOD*, Seattle, Washington, 1998.

[32] Edwin K. P. Chong, Robert L. Givan, and Hyeong Soo Chang. A framework for simulation-based network control via hindsight optimization. In *Proc. of the 39th IEEE Conf. on Decision and Control*, Sydney, Australia, December 2000.

[33] Mikkel Christiansen, Kevin Jaffay, David Ott, and F. Donelson Smith. Tuning RED for web traffic. In *Proceedings of ACM SIGCOMM '2000*, pages 139–150, 2000.

[34] Dave Clark, Bruce S. Davie, and Larry L. Peterson. *Computer Networking: A Systems Approach*. Morgan Kaufmann Publishers, 1996.

[35] Fred Cohen, Dave Lambert, Charles Preston, Nina Berry, Corbin Stewart, and Eric Thomas. A framework for deception. http://heat.ca.sandia.gov/papers/Framework/Framework.html, July 2001. Technical Baseline Report.

[36] James Cowie, David M. Nicol, and Andy T. Ogielski. Modeling the global internet. *Computing in Science & Engineering*, 1(1):42–50, January/February 1999.

[37] R. L. Cruz. A calculus for network delay, part i: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114 –131, January 1991.

[38] Joanne Cummings. From intrusion detection to intrusion prevention. In *NetworkWorld*. Network World Inc., 118 Turnpike Road, Southborough, MA 01772, September 2002.

[39] Joanne Cummings. A security godsend. In *NetworkWorld*. Network World Inc., 118 Turnpike Road, Southborough, MA 01772, December 2002.

[40] A.J. David and B.E.A. Saleh. Optical implementation of the hopfield algorithm using correlations. *Applied Optics*, 29(8):1063–1064, March 1990.

[41] B. D. Davison and H. Hirsh. Naive bayes appropriate user actions. In *Predicting the Future: AI Approaches to Time-Series Problems,*, pages 5–12, Menlo Park, California, July 1998. AAAI Press, Madison, Wisconsin. Papers from the 1998 AAAI Workshop.

[42] J. Day and H. Zimmerman. The OSI reference model. In *Proceedings of the IEEE*, volume 71, pages 1334–1340, December 1983.

[43] Paul Desmond. IDS tools smarten up. In *NetworkWorld*. Network World Inc., 118 Turnpike Road, Southborough, MA 01772, September 2002.

[44] Peter V. DeSouza. Statistical test and distance measures for lpc coefficients. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-25(6):554–558, December 1977.

[45] Distributed Management Task Force, Inc. Common information model (cim) specification. http://www.dmtf.org/standards/documents/CIM/DSP0004.pdf, June 1999.

[46] Y. Dong, X. Du, and X. Zhang. Characterizing and scheduling communication interactions of parallel and local jobs on networks of workstations. *Computer Communications*, 21(5):470–484, 1998.

[47] Xing Du, Yingfei Dong, and Xiaodong Zhang. Characterizing communication interactions of parallel and sequential jobs on networks of workstations. In *ICC (2)*, pages 1133–1137, 1997.

[48] William DuMouchel. Computer intrusion detection based on bayes factors for comparing command transition probabilities. Technical Report TR91, National Institute of Statistical Sciences (NISS), 1999.

[49] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry. The cops (common open policy service) protocol. Request for Comments: 2748, January 2000.

[50] David Durham. A new paradigm for policy-based network control. *Intel Developer Update Magazine*, November 2001.

[51] S.T. Eckmann, G. Vigna, and R.A. Kemmerer. Statl: An attack language for state-based intrusion detection. In *Proceedings of the ACM Workshop on Intrusion Detection*, Athens, Greece, November 2000.

[52] Simon Edwards. Network intrusion detection systems: Important ids network security vulnerabilities – must know information on realizing and overcoming the risks. http://www.toplayer.com/, September 2002.

[53] Laurent Eschenauer. ImSafe - host based anomaly detection tool. http://imsafe.sourceforge.net/, February 2001.

[54] N.Y. Farhat, D. Psaltis, A. Prata, and E. Paek. Optical implementation of the hopfield model. *Applied Optics*, 29(8):1469–1475, 1985.

[55] Anja Feldmann, Anna C. Gilbert, Polly Huang, and Walter Willinger. Dynamics of IP traffic: A study of the role of variability and the impact of control. In *SIGCOMM*, pages 301–313, 1999.

[56] Wu-chang Feng. *Improving Internet Congestion Control and Queue Management Algorithms*. PhD thesis, The University of Michigan, Ann Harbor, Michigan, 1999. Computer Science and Engineering.

[57] Sally Floyd. A report on some recent developments in TCP congestion control. *IEEE Communications Magazine*, April 2001. http://www.aciri.org/floyd/papers/report_Jan01.pdf.

[58] Sally Floyd. References on RED (Random Early Detection) queue management. http://www.icir.org/floyd/red.html, 2002.

[59] Sally Floyd and K. K. Ramakrishnan. ECN (explicit congestion notification) in TCP/IP. http://www.aciri.org/floyd/ecn.html, May 2001.

[60] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for Unix processes. In *Proceedinges of the 1996 IEEE Symposium on Research in Security and Privacy*, pages 120–128. IEEE Computer Society Press, 1996.

[61] Svend Frølund and Pankaj Garg. Design-time simulation of a large-scale, distributed object system. *ACM Transactions on Computer Modeling and Simulation*, 8(4):374 – 400, October 1998. Special issue on Web-based modeling and simulation.

[62] Svend Frølund, Pankaj Garg, and Allan Shepherd. On predicting the performance and scalability of distributed applications. In *Proceedings of the International Symposium on Software Engineering for the Next Generation*, Nagoya, Japan, February 1996. Nanzan Institute of Business Administration.

[63] Alfonso Fuggetta, Gian Pietro Picco, and Giovanni Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, May 1998.

[64] Thomas D. Garvey and Teresa F. Lunt. Model-based intrusion detection. In *Proceedings of the 14th National Computer Security Conference*, pages 405–418, October 1991.

[65] Anup K. Ghosh, Aaron Schwartzbard, and Michael Schatz. Learning program behavior profiles for intrusion detection. In *Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring*, Santa Clara, CA., April 1999. Reliable Software Technologies Corporation.

[66] Lee Giles and Mark Goudreau. Routing in optical multistage interconnection networks: a neural network solution. *Journal of Lightwave Technology*, 13(6):1111–1115, June 1995.

[67] L. Girardin and D. Brodbeck. A visual approach or monitoring logs. In *Proceedings of the* 12^{th} *System Administration Conference (LISA '98)*, volume 11, pages 299–308, Boston, MA, December 1998.

[68] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Reading, MA, 1989.

[69] Duncan Graham-Rowe. It's bacteria's turn to shine. *New Scientist/RBI Magazines Ltd.*, October 2000. http://www.newscientist.com/news/news.jsp?id=ns225939.

[70] Ananth Grama, Anshul Gupta, and Vipin Kumar. Isoefficiency. measuring the scalabbility of parallel algorithms and architectures. *IEEE Parallel & Distributed Technology*, 1(3):12–21, Aug. 1993.

[71] Seth Grimes. Autonomic computing: Major vendors are applying decision-support techniques to service-centric computing. Intelligent Enterprise, November 2002. http://www.intelligententerprise.com/021115/518decision1_2.shtml.

[72] Neil J. Gunther. *The Practical Performance Analyst: performance-by-design techniques for distributed systems.* McGraw-Hill Series on Computer Communications. McGraw-Hill, New York, NY, 1998.

[73] J. R. Harista, M. O. Ball, N. Roussopoulos, A. Datta, and J. S. Baras. MANDATE: MAnaging Networks using DAtabase TEchnology. *IEEE Journal on Selected Areas in Communications*, 11(9):1360–1372, December 1993.

[74] David Harrison and Shivkumar Kalyanaraman. Edge-to-edge traffic control: A new overlay congestion control architecture for the internet. Technical report, Rensselaer Polytechnic Institute, Troy, New York, March 2000. http://poisson.ecse.rpi.edu/ harrisod/.

[75] Simon Haykin. *Neural Networks.* Prentice Hall Inc., Upper Saddle River, New Jersey, second edition, 1999.

[76] L. Todd Heiberlein, Gihan V. Dias, Karl N. Levit, Biswanath Mukherjee, Jeff Wood, and David Wolber. A network security monitor. In *Proceedings of the 1990 Symposium on Research in Security and Privacy*, pages 296–304, Oakland, CA, May 1990. IEEE.

[77] Hewlett-Packard. hp OpenView extensible SNMP agent product brief. http://www.openview.hp.com/products/extensible_snmp_agent/briefs, June 2001.

[78] Hewlett-Packard. hp OpenView performance manager and performance monitor product brief. http://www.openview.hp.com/products/performance/briefs, May 2002.

[79] Paul Horn. *Autonomic Computing: IBM's Perspective on the State of Information Technology.* International Business Machines Corporation, New Orchard Road, Armonk, NY 10504, October 2001.

[80] Paul Horn. How autonomic computing will reshape it. CNet News.com, October 2001. http://news.com.com/2010-1079-281578.html?legacy=cnet.

[81] Bradley Huffaker, Marina Fomenkov, Daniel J. Plummer, David Moore, and k claffy. Distance metrics in the internet. In *IEEE International, Telecommunications Symposium, (ITS2002)*, Brazil, September 2002. IEEE.

[82] Christian Huitema. *Routing in the Internet*. Prentice Hall PTR, Upper Saddle River, NJ 07458, second edition, 2000.

[83] IBM. IBM tivoli monitoring for network performance. http://www.tivoli.com/products/index/monitor-net-performance/, October 2002.

[84] IBM Research. Aglets software development kit. http://www.trl.ibm.com/aglets/, 1998.

[85] Koral Ilgun, Richard A. Kemmerer, and Phillip A. Porras. State transition analysis: A rule-based intrusion detection approach. *IEEE Transactions on Software Engineering*, 21(3):181–199, 1995.

[86] Internet Security Systems. ISS BlackICE™PC Protection product spec sheet. http://blackice.iss.net/.

[87] Internet Security Systems. ISS RealSecure® product spec sheet. http://www.iss.com/.

[88] V. Jacobson, C. Leres, and S McCanne. Tcpdump, June 1989. Available via anonymous FTP from ftp.ee.lbl.gov.

[89] R. Jain. A delay based approach for congestion avoidance in interconnected heterogeneous computer networks. *Computer Communications Review, ACM SIGCOMM*, pages 56–71, 1989.

[90] Raj Jain, Shivkumar Kalyanaraman, Sonia Fahmy, Rohit Goyal, and Seong-Cheol Kim. Tutorial paper on ABR source behavior. ATM Forum/96-1270, October 1996. http://www.cis.ohio-state.edu/ jain/atmf/a96-1270.htm.

[91] Gary Jones. Objectstore 6.0. http://www.objectdesign.com/htm/object_prod.htm, October 1999. product white paper.

[92] Y. Joo, V. Ribeiro, A. Feldmann, A.C. Gilbert, and W.Willinger. TCP/IP traffic dynamics and network performance: A lesson in workload modeling, flow control, and trace-driven simulations. *SIGCOMM Computer Communications Review*, 31(2), April 2001.

[93] W. Ju and Y. Vardi. A hybrid high-order markov chain model for computer intrusion detection. Technical Report 92, National Institute for Statistical Sciences, Research Triangle Park, North Carolina 27709-4006, 1999.

[94] Peter Judge. Hp recycles old tech for new data centres. ZD Net, February 2002. http://news.zdnet.co.uk/story/0,,t269-s2105199,00.html.

[95] Michael J. Katchabaw, Stephen L. Howard, Hanan L. Lutfiyya, and Micheal A. Bauer. Efficient management data acquisition and run-time control of dce applications using the osi management framework. In *the Proceedings of the IEEE Second International Workshop on Systems Management*, pages 104–111, Toronto, Canada, June 19-21 1996.

[96] Kris Kendall. A database of computer attacks for the evaluation of intrusion detection systems. Master's thesis, Massachusetts Institute of Technology, 1998.

[97] Gene Kim. Advanced applications of tripwire for servers: Detecting intrusions, rootkits, and more. http://www.tripwire.com/.

[98] Leonard Kleinrock. *Queueing Systems: Volume II: Computer Applications*, volume II. John Wiley & Sons, Inc, New York, NY, 1976.

[99] T. Kohonen. *Self-Organizing Maps*, volume 3. Springer-Verlang, Springer-Verlang Berlin, 1995.

[100] Sri Kumar. Network modeling and simulation. DARPA Case no. 39152, April 2002. Network Modeling and Simulation Principal Investigators' Meeting.

[101] V. P. Kumar and A. Gupta. Analyzing scalability of parallel algorithms and architectures. *Journal of Parallel and Distributed Computing*, 22(3):379–391, 1994.

[102] Věra Kůrková. *The Handbook of Brain Theory and Neural Networks*, chapter Kolmogorov's theorem, pages 501–502. MIT Press, Cambridge, Massachusetts, 1995.

[103] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet.* Addison Wesley. Pearson Education, Inc., 2 edition, 2003.

[104] T. Lane and C. E. Brodley. Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security*, 2(3):295–331, August 1999.

[105] Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632, 1984.

[106] Yihua Liao and V. Rao Vemuri. Use of k-nearest neighbor classifier for intrusion detection. *Computers & Security*, 21(5):439–448, October 2002.

[107] Yihua Liao and V. Rao Vemuri. Using text categorization techniques for intrusion detection. In *Proceedings of the 11th USENIX Security Symposium*, pages 51–59, San Francisco, CA, August 2002.

[108] Dong Lin and Robert Morris. Dynamics of random early detection. In *Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 127–137, Cannes, France, September 1997. ACM SIGCOMM.

[109] Richard Lippmann, Robert K. Cunningham, David J. Fried, Isaac Graf, Kris R. Kendall, Seth E. Webster, and Marc A. Zissman. Results of the darpa 1998 offline intrusion detection evaluation. In *Recent Advances in Intrusion Detection 1999*, West Lafayette, Indiana, September 1999. Second International Workshop, RAID 1999.

[110] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. The 1999 darpa off-line intrusion detection evaluation. *Computer Networks*, 34(4):579–595, October 2000.

[111] Richard P. Lippmann and Robert K. Cunningham. Using key-string selection and neural networks to reduce false alarms and detect new attacks with sniffer-based intrusion detection systems. In *RAID 1999 Conference*, West Lafayette, Indiana, September 1999.

[112] G. Mansfield, E. P. Duarte Jr., M. Kitahashi, and S. Noguchi. Vines: Distributed algorithms for a web-based distributed network management system. In *Worldwide Computing and Its Applications*, pages 281–293, Tsukuba, Japan, March 10-11 1997. International Conference, WWCA, Springer Verlag.

[113] David Marchette. A statistical method for profiling network traffic. In *Proceedings of the Workshop on Intrusion Detection and Network Monitoring*, pages 119–128, 1999.

[114] Roy A. Maxion and Tahlia N. Townsend. Masquerade detection using truncated command lines. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN-02)*, pages 219–228, Washington, D.C., June 2002. IEEE Computer Society Press, Los Alamitos, California.

[115] K. McCloghrie and M. Rose. Management information base for network management of TCP/IP-based internets: MIB-II. Request for Comments: 1213, March 1991. Obsoletes: RFC 1158.

[116] S. McCloghrie and J. Scambray. Once-promised intrusion detection system stumbles over switched networks. In *InfoWorld*, volume 22, page 58. InfoWorld Media Group, Inc., December 2000.

[117] Stuart McClure and Joel Scambray. Once-promising intrusion detection systems stumble over switched networks. In *InfoWorld*, volume 22, pages 58–58. InfoWorld Media Group, Inc., December 2000.

[118] Ellen Messmer. Companies target the enemy within. In *NetworkWorld*. Network World Inc., 118 Turnpike Road, Southborough, MA 01772, August 2002.

[119] Ellen Messmer. IntruVert inspects high-speed ip traffic. In *NetworkWorld*. Network World Inc., 118 Turnpike Road, Southborough, MA 01772, September 2002.

[120] Mirza Murtaza and Jayneen Shah. Reducing data movement in client/server systems using neural networks. http://www.informs.org/Conf/NO95/TALKS/MA19.3.html, November 1995.

[121] John Nagle. On packet switches with infinite storage. *IEEE Transactions on Communications*, COM-35(4):435–438, April 1987.

[122] David Newman, Joel Snyder, and Rodney Thayer. Network-based intrusion-detection systems crying wolf: False alarms hide attacks. In *NetworkWorld*, volume 19, pages 57–64. Network World Inc., 118 Turnpike Road, Southborough, MA 01772, June 2002.

[123] NFR Security. NFR host intrusion detection v2.0 data sheet. http://www.nfr.com/products/HID/docs/NFR_HID_Data_Sheet.pdf.

[124] NFR Security. NFR network intrusion detection v2.0 data sheet. http://www.nfr.com/products/NID/docs/NFR_NID_Data_Sheet.pdf.

[125] O. Niggemann, B. Stein, and J. Tlle. Visualization of traffic structures. In *IEEE International Conference on Communications*, volume 5, pages 1516 –1521, 2001.

[126] Richard Ogier, Nina T. Plotkin, and Irfan Khan. Neural network methods with traffic descriptor compression for call admission control. In *Proceedings of the Conference on Computer Communications*, San Francisco, California, March 1996. IEEE Infocom.

[127] David Pescovitz. Helping computers help themselves. *IEEE Spectrum*, 51:49–53, September 2002.

[128] P.A. Porras and P. Neumann. Emerald: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings 20th National Information Systems Security Conference*, pages 353–365, Baltimore, Maryland, October 1997.

[129] Leonid Portnoy, Eleazar Eskin, and Willian Salvatore J. Stolfo. Intrusion detection with unlabeled data using clustering. *To Appear in Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001)*, 2001.

[130] J. Postel. User datagram protocol. Request for Comments: 768, August 1980.

[131] Aiko Pras, Bert-Jan van Beijnum, and Ron Sprenkels. Introduction to tmn. Technical Report TR-CTIT 99-09, Centre for Telematics and Information Technology (CTIT), University of Tente, Netherlands, April 1999.

[132] ProductivityNet. ActiveManage^{TM}version 2.0 white paper. http://www.productivitynet.com/activemanagewhitepaper.pdf, 2002.

[133] K. Ramakrishnan and S. Floyd. A proposal to add explicit congestion notification (ECN) to IP. Request for Comments: 2481, January 1999. ftp://ftp.isi.edu/in-notes/rfc2481.txt.

[134] Vivek Rao, Saurabh Gupta, and Milan Parmar. Mobile agent framework. available at www.cs.rpi.edu/~szymansk/doors/papers/Agent.doc, March 2000.

[135] Recourse Technologies. ManHunt®2.1 reducing the risk of compromise. http://enterprisesecurity.symantec.com/.

[136] Recourse Technologies. ManTrap®3.0 early warning systems for advanced protection. http://enterprisesecurity.symantec.com/.

[137] Martin Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of the LISA'99 conference. USENIX*. USENIX, November 1999. URL http: //www.snort.org/lisapaper.txt.

[138] Jake Ryan, Meng-Jang Lin, and Risto Miikkulainen. Intrusion detection with neural networks. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

[139] Stefan Savage, Neal Cardwell, David Wetherall, and Tom Anderson. TCP congestion control with a misbehaving receiver. Technical Newsletter: Computer Communication Review, October 1999.

[140] Erick Schonfeld. Computing to the Nth degree. Business 2.0, September 2002. http://www.business2.com/articles/mag/0,1640,42876,00.html.

[141] M. Schonlau and M. Theus. Detecting masquerades in intrusion detection based on unpopular commands. *Information Processing Letters*, 76(1–2):33–38, November 2000.

[142] Matthias Schonlau, William DuMouchel, Wen-Hua Ju, Alan F. Karr, Martin Theus, and Yehuda Vardi. Computer intrusion: Detecting masquerades. *Statistical Science*, 16(1):58–74, February 2001.

[143] M. Schwartz. Performance analysis of the SNA virtual route pacing control. *IEEE Transactions on Communications*, COM-30(1):172–184, January 1982.

[144] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.

[145] Stephen E. Smaha. Haystack: An intrusion detection system. In *Proceedings of the Fourth Aerospace Computer Security Applications Conference*, pages 37–44, Orlando, Florida, December 1988.

[146] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, S3A-13(147):195–197, 1981.

[147] William Stallings. *SNMP, SNMPv2, SNMPv3, and RMON1 and 2*. Addison Wesley Longman, Inc., Reading, Massachusetts, 3rd edition, 1999.

[148] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS – A graph-based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference*, volume 1, pages 361–370, October 1996.

[149] Henry Steinhauer. Network Traffic Analysis at the 20,000 Foot Level – or where did all this traffic come from. http://people.ee.ethz.ch/ oetiker/webtools/mrtg/pub/papers/steinhauer98.pdf, 1998.

[150] W. Stevens. TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. Request for Comments: 2001, January 1997. Category: Standards Track.

[151] W. Richard Stevens. *TCP/IP Illustrated Volume 1: The protocols*, volume 1. Addison-Wesley Publishing Company, One Jacob Way; Reading, Massachusetts 01867, 1994.

[152] W. Richard Stevens. *TCP/IP Illustrated*, volume 3. Addison-Wesley Publishing Company, One Jacob Way; Reading, Massachusetts 01867, 1996.

[153] Rajesh Subramanyan, José Miguel-Alonso, and José A. B. Fortes. Design and evaluation of a snmp-based monitoring system for heterogeneous, distributed computing. Technical Report TR-ECE 00-11, School of Electrical and Computer Engineering, Purdue Universit, West Lafayette, IN, July 2000.

[154] Rajesh Subramanyan, José Miguel-Alonso, and José A. B. Fortes. A scalable SNMP-based distributed monitoring system for heterogeneous network computing. In *the Proceedings of Supercomputing2000*, pages 52–52, Dallas, TX, November 4-10 2000.

[155] Sun Microsystems, Inc. Answerbook2: Sunshield basic security module guide. http://docsun.cso.uiuc.edu/cgi-bin/nph-dweb/ab2/coll.47.11/SHIELD/@Ab2PageView/3507, 2000.

[156] SYCHRON INC. Autonomic computing with sychron software. http://www.sychron.com/autonomics/index.html, 2001.

[157] Symantec. NetProwler™dynamic intrusion detection for enterprise networks (fact sheet). http://enterprisesecurity.symantec.com/.

[158] Symantec. Symantec™host IDS: Automated intrusion detection and response for systems, applications, and data (fact sheet). http://enterprisesecurity.symantec.com/.

[159] B. Szymanski. Genesis: Network monitoring and simulation for network management. Congress of Informatics, June 2003. to appear.

[160] B. Szymanski, Q. Gu, and Y. Liu. Time-network partitioning for large-scale parallel network simulation under ssfnet. In *Proceedings of the Applied Telecommunication Symposium*, pages 90–95, San Diego, CA, April 2002. SCS Press.

[161] B. Szymanski, Y. Liu, A. Sastry, and K. Madnani. Real-time on-line network simulation. In *Proceedings of the 5th IEEE International Workshop on Distributed Simulation and Real-Time Applications DS-RT 2001*, pages 22–29, Cincinnati, OH, August 2001. IEEE Computer Society Press, Los Alamitos, CA.

[162] B. Szymanski, A. Saifee, A. Sastry, Y. Liu, and K. Mandani. Genesis: A system for large-scale parallel network simulation. In *Proceedings of the 16th Workshop on Parallel and Distributed Simulation*, pages 89–96, Washington, DC, May 2002. IEEE CS Press.

[163] Andrew Tanenbaum. *Computer Networks*. Prentice Hall PTR, Upper Saddle River, New Jersey, third edition edition, 1996.

[164] The Ipswitch Service & Support Teams. Whatsup gold - spurious down reports from snmp services such as interface. http://support.ipswitch.com/kb/WG-20011221-DM01.htm, February 2002. The Ipswitch Knowledge Base.

[165] David L. Tennenhouse and David J. Wetherall. Towards an active network architecture. Technical Newsletter: Computer Communication Review, April 1996.

[166] C.K. Tham and W. S. Soh. Multi-service connection admission control using modular neural networks. In *Proceedings of the Conference on Computer Communications*, page 1022, San Francisco, California, March 1998. IEEE Infocom.

[167] Marina Thottan and Chuanyi Ji. Adaptive thresholding for proactive network problem detection. In *IEEE Internation Workshop on Systems Management*, pages 108–116, Newport, Rhode Island, April 1998. IEEE.

[168] Marina Thottan and Chuanyi Ji. Proactive anomaly detection using distributed intelligent agents. *IEEE Network, Special Issue on Network Management*, 12(5):21 –27, Sept-Oct 1998.

[169] Greg Todd. What is WBEM? In *Windows & .Net Magazine*. Penton Media, Inc., July 1998.
http://www.win2000mag.com/Articles/Index.cfm?ArticleID=3568.

[170] Ljiljana Trajkovic and Arnie Neidhardt. Effect of traffic knowledge on the efficiency of admission-control policies. Technical Newsletter: Computer Communication Review, January 1999.

[171] Denish Verma. *Supporting Service Level Agreements on IP Networks.* MacMillan Technology Series. Macmillan Technical Publishing, 201 West 103rd Street, Indianapolis, IN 46290, 1999.

[172] J. Vesanto and E. Alhoniemi. Clustering of the selforganizing map. *IEEE Transactions on Neural Networks*, 11(3):586–600, May 2000.

[173] Jeffrey S. Vetter. Dynamic statistical profiling of communication activity in distributed applications. In *Proceedings of SIGMETRICS 2002, International Conference on Measurements and Modeling of Computer Systems*, pages 240–250, Marina Del Rey, California, June 2002. ACM.

[174] Giovanni Vigna and Richard A. Kemmerer. Netstat: A network-based intrusion detection approach. In *Proceedings of the 14th Annual Computer Security Conference*, pages 25–36, Scottsdale, Arizona, December 1998.

[175] Bob Violino. Fortifying the firewall. In *NetworkWorld*. Network World Inc., 118 Turnpike Road, Southborough, MA 01772, July 2002.

[176] S. Waldbusser. Remote network monitoring management information base. Request for Comments: 1757, February 1995.

[177] Christina Warrender, Stephanie Forrest, and Barak A. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on Security and Privacy*, pages 133–145, 1999.

[178] U. Warrier, L. Besaw, L. LaBarre, and B. Handspicker. The common management information services and protocols for the internet (cmot and cmip). Request for Comments: 1189, October 1990.

[179] Seth E. Webster. The development and analysis of intrusion detection algorithms. Master's thesis, MIT, 1998.

[180] Todd R. Weiss. Self-learning comes to ibm's eliza self-healing initiative. ComputerWorld, May 2002.

[181] T. A. Welch. A technique for high performance data compression. *IEEE Computer Magazine*, 17(6):8–19, June 1984.

[182] Tao Ye, Shivkumar Kalyanaraman, David Harrison, Biplap Sikdar, Bin Mo, Hema Tahilramani Kaur, Ken Vastola, and Boleslaw Szymanski. Network management and control using collaborative on-line simulation. In *IEEE International Conference on Communications ICC2001*, Helsinki, Finland, June 2001.

[183] A. Ypma and R. Duin. Novelty detection using self-organizing maps. *Progress in Connectionist-Based Information Systems*, 2:1322–1325, 1997.

[184] Jian Yuan and Kevin Mills. Exploring collective dynamics in communication networks. *Journal of Research of the National Institute of Standards and Technology*, 107(2):179–191, March–April 2002.

[185] M. Yuksel, N. Sikdar, B. Szymanski, and K. Vastola. Workload generation for *ns* simulations of wide area networks and the internet. In *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation*, pages 93–98, San Diego, CA, 2000. SCS Press.

[186] William Yurcik, David Doss, Hans Kruse, and J. Warren McClure. Challenges to the end-to-end internet model. In *Americas Conference on Information Systems*, Long Beach, CA, August 2000. AMCIS 2000.

[187] Ji-Fang Zhang, Jingjie Jiang, and Boleslaw K. Szymanski. A distributed simulator for large-scale networks with on-line collaborative simulators. In *Proc. European Multisimulation Conference*, volume II, pages 146–150, Brussels, Belgium, June 1999. Society for Computer Simulation Press. Warsaw, Poland.

[188] Xiaodong Zhang, Yong Yan, and Keqiang He. Latency metric: and experimental method for measuring and evaluating parallel program and

architecture scalability. *Journal of Parallel and Distributed Computing*, 22(3):392–410, September 1994.

[189] Laurie Zirkle. What is host-based intrusion detection? In *SANS Institute Resources: Intrusion Detection FAQ*. SysAdmin, Audit, Network, Security Institute, 2000.