WinNTGen: Creation of a Windows NT 5.0+ Network Traffic Generator

by

Jesse C. Boothe-Rabek

B.S. Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 2001

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2003

Signature of Author:____________ ________
Department of Electrical Engineering and Computer Science
January 18, 2003

Certified by:______________________
Robert K. Cunningham
Assistant Group Leader, MIT Lincoln Laboratory
Thesis Supervisor

Accepted by:_________ ______________________________
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

WinNTGen: Creation of a Windows NT 5.0+ Network Traffic Generator

By
Jesse C. Boothe-Rabek

Submitted to the Department of Electrical Engineering and Computer Science

February 18, 2003

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The Lincoln Adaptable Real Time Information Assurance Testbed (LARIAT) project is the first fully automatable network testbed for the evaluation of information assurance (IA) technologies. It allows researchers to easily set up experiments that evaluate the accuracy of host-based and network-based intrusion detection systems (IDSs). Initially, the network traffic it could produce used UNIX services and protocols as implemented for the Linux and Solaris platforms. However, due to the widespread deployment of Windows-based systems in production environments, it is necessary to include Windows-based traffic when testing IA systems in order to provide a comprehensive evaluation.

This thesis describes WinNTGen, a Windows network traffic generation system that integrates into the existing LARIAT framework and enables it to produce Windows-based network traffic. To do this, WinNTGen simulates the actions of a user controlling applications that in turn use network resources. This frees WinNTGen from the need to re-implement network protocols and allows it to operate at a higher level of abstraction.

WinNTGen controls applications via loadable libraries that encapsulate the manner in which a typical user interacts with a particular application. The statistical parameters that specify the behavior of a user with each application are derived from real users' behavioral data as they interacted with each application. The system is flexible and extensible so that different versions of the same application as well as additional applications can be controlled by modifying and adding libraries.

Finally, the reality and the throughput of the network traffic produced by the WinNTGen system are evaluated.

Thesis Supervisor: Robert K. Cunningham
Title: Assistant Group Leader, MIT Lincoln Laboratory

Acknowledgements

There are many people that helped me along the way to completing this thesis and I do not think I could possibly name them all. First and foremost I would like to thank my thesis supervisor, Rob Cunningham, for his excellent insight, assistance, and encouragement throughout. Also, I would like to thank Lee Rossey and Gerry O'Leary (aka the LARIAT team) for countless hours of discussion and advice.

During the entire project, the entire Group 62 staff at Lincoln has been great in helping me. For the user modeling, Rich Lippmann, Doug Reynolds, and Oliver Dain provided wonderful expertise and analysis. For every technical aspect of Windows NT, Scott Lewandoski was a source of invaluable information.

For moral support I want to thank my family, Norman Rabek, Joy Boothe, and Eli Rabek; my girlfriend, Ana Rodrigo; and, of course, all of my brothers at Nu Delta Fraternity.

Contents

Index of Figures

Index of Tables

Chapter 1

Introduction

1.1 Evaluating Network-Based Information Assurance Systems

Due to the rapid growth of technologies surrounding networked computing resources, it has become necessary to create fully automated test systems for evaluating network-based Information Assurance (IA) systems. In a general sense, IA is "knowledge management"[1]. Currently IA technologies are "most closely associated with detection of and response to vulnerabilities and events relating to cyber attacks"[1] although the area of IA covers many other technologies such as authentication and cryptology. More formally, IA technologies are responsible for protecting data, detecting unauthorized manipulations of data, and responding appropriately to any breaches of security.

There is a large need for comprehensive IA systems. Annual losses by corporations, universities, and government agencies due to computer crimes is measured in billions of dollars and many of the attacks go unnoticed or unreported[2]. These attacks can come from both inside and outside of organizations.

One of the rapidly developing information assurance technologies that warrants focus is the intrusion detection system (IDS). IDSs attempt to signal system misuse through examination of network traces, audit logs from individual hosts, or both. Even though there are technologies to limit abuse, such as firewalls, these systems are fallible since they have incomplete knowledge and are static in nature. Because computer networks and the applications that communicate across them are complex and rapidly changing, bugs, flaws, and weaknesses are constantly introduced that could allow a malicious user to gain access to private resources. It is therefore necessary that these systems be deployed alongside a more sophisticated IDS capable of alerting system administrators to possible abuses so that vulnerabilities can be eliminated.

In order to evaluate the accuracy of the technical approach of a particular IDS, it is necessary to create a controlled network environment in which the IDS examines network traffic and host logs produced in a predetermined manner. This way, specific aspects of the IDS may be tested by comparing IDS alerts to ground truth. IDS evaluations have been conducted in this manner by several groups. A summary of some of the more widely known evaluations can be found in *Network Intrusion Detection: An Analyst's Handbook*[3].

The DARPA 1998[4] and 1999[5, 6] offline intrusion detection evaluations conducted at the Massachusetts Institute of Technology's Lincoln Laboratory provided assessments of various IDSs by supplying the IDS researchers with network traces and host log data that had been recorded from a network testbed. Researchers reported the alerts generated by their IDSs, and these were scored using a list of when the attacks actually occurred. In the 1998 evaluation only UNIX style background traffic and attacks were included. The 1999 evaluation included all the traffic types found in the 1998 evaluation as well as a limited variety of network traffic produced by Windows NT 4.0 that was mostly produced by more traditional protocols such as HTTP. Attacks in both evaluations were executed by a human actor.

The Lincoln Adaptable Real-time Information Assurance Testbed (LARIAT) [7, 8] was developed to address certain limitations and inconveniencies of the evaluations and its tool set by automating the time consuming tasks of setting up the evaluation run and collecting run-time statistics and data. By automating these tasks and allowing for multiple network configurations, organizations using LARIAT are able to perform experiments tailored to their specific network environment. LARIAT emphasizes automatically generating repeatable real time background and attack traffic for evaluating IA systems. This also allows LARIAT to be used in the evaluation of host-based IDSs, which requires real time network traffic. Initially LARIAT used

UNIX services and protocols as implemented for the Linux platform in order to produce the network traffic. Attacks in LARIAT are completely automated and it is possible to chain together multi-stage attacks. Recent work on the Thor system [9] built upon the automated attack ideas in LARIAT and added the ability to vary individual attacks.

Given the facts that LARIAT was not initially capable of producing extensive network traffic from Windows NT, and that and it is common for a network to include hosts running the Windows NT OS, as can be seen from Section 2.4, it was necessary to create a system capable of producing Windows NT traffic that could integrate with LARIAT.

1.2 The Development of a Windows Traffic Generator

This thesis describes the design and implementation of WinNTGen, a system that produces realistic Windows NT network traffic and integrates into the existing LARIAT system. The system generates traffic associated with Microsoft implementations of cross-platform services, such as HTTP traffic, as well as Microsoft-specific traffic that results from communication between Windows Workstations and Windows Servers, such as SMB over TCP/IP.

1.3 Thesis Outline

Chapter 2 describes the previous work in the area of traffic generation and evaluation of information assurance technologies, as well as the motivation for creating a Windows traffic generation system. Chapter 3 covers the way in which Windows NT users are measured and modeled and how these models are used to produce network traffic. Chapter 4 provides implementation details of the WinNTGen system, several attempted and rejected designs, and a description of how WinNTGen is integrated into LARIAT. Chapter 5 evaluates the performance of the system in terms of traffic throughput and resemblance to actual traffic. Finally, Chapter 6 draws conclusions about WinNTGen and describes future directions it could take.

Chapter 2

Motivation and Background

2.1 Motivation

Given the potential of large financial losses [2] and other immeasurable damages caused by compromised information as a result of a cyber attack, it is desirable for institutions to deploy IA systems capable of protecting themselves from such threats. However, if the IA system itself is flawed it only serves to provide a false sense of security which is certainly worse than having no system at all. It is therefore necessary to evaluate such systems in order to determine their true effectiveness. Network-based and host-based IA technologies, specifically IDSs, are evaluated by providing them with network traffic, host log data, or both, and then examining the alerts generated in order to determine if the IDS did indeed detect everything it should have.

There is a temptation for an institution interested in deploying an IDS to perform an evaluation of several IDSs on their own network to determine which one performs best in their particular network environment. However, this quickly becomes an expensive proposition since time must be spent in installing and configuring each IDS in their network environment as well as evaluating experiment results. A recent IDS evaluation performed by Network World took three months, involved three full time staff, and cost tens of thousands of dollars[10]. For an institution evaluating a small subset of IDSs for their own use, there is no guarantee that the IDSs that they choose to evaluate perform the best in their type of network environment.

A more viable, cost effective approach to the IDS evaluation problem is to use a network testbed that is designed for rapid reconfiguration and testing of IDSs. Using a testbed approach there are several possible ways in which an IDS can be evaluated.

The first possibility is that vendors can be provided with a testbed environment with which they can evaluate their own IDSs before they are deployed. The advantage to this approach is that it allows vendors to shorten their evaluation and improvement cycle as well as perform thorough evaluations of their final product before it is released. However, this does have the disadvantage that vendors could create experiments that make their IDS appear to perform better than it actually does. In order to address concerns about inaccurate evaluations by vendors, there is another possibility that a centralized IDS evaluation can be performed by an independent group in which multiple IDSs are evaluated on several common network configurations. Organizations could then take the two or three IDSs that performed the best in the network configuration that most closely resembled their own and, if they choose, perform further evaluations on this subset in order to select the best one of these to use.

Section 2.2 gives an overview of the 1998 and 1999 DARPA IDS evaluation which is the original research that uses a variety of testbed generated network traffic as a part of IDS evaluation. Section 2.3 describes LARIAT which grew out DARPA IDS evaluation and sought to address some of the most important limitations of the previous system.

2.2 1998 & 1999 DARPA IDS Evaluation

The IDS evaluation conducted at the Massachusetts Institute of Technology's Lincoln Laboratory [6] is the first to provide a system to evaluate the technical accuracy of DARPA funded IDSs. The research built upon previous approaches to evaluating IDSs [11-13], substantially increasing the number of attacks and expanding the breadth of background traffic. Its purpose was to provide rich data sets for testing and experimentation with various ID algorithms and technologies, to evaluate ID approaches by analyzing the strengths and weaknesses of each, to facilitate analysis of how and why alternate approaches to IDSs differ, and to report to DARPA information necessary to guide and focus future research directions[6].

The authors indicate that future evaluations should include more Windows NT traffic and that the traffic should be scriptable. The first part of the statement is motivated by the fact that the Windows OS is widely deployed, and should be included in a comprehensive IDS evaluation. The second part of the statement was made because in the 1999 DARPA IDS Evaluation during the traffic generation sessions, human actors would perform on the order of one to three manual actions each day. These tasks included, but were not limited to, upgrading software, adding users, and changing passwords. As a result, the recorded audit data could no be perfectly recreated. Scripting these actions would add more accurate simulation repeatability and better ground truth.

Also relevant to this thesis is the fact that in the 1999 DARPA IDS Evaluation network traffic produced by the Windows NT machines was limited in scope in that many of the Windows NT-specific protocols were not used. Specifically, in that evaluation there were three different types of Windows NT background traffic present. The first was from telnet connections to a Windows NT host made by simulated users from RedHat Linux hosts. The second was from an auto-browser module residing on the Windows NT host that simulated a user browsing to different predefined web pages. The auto-browser was built using JavaScript and ran out of Netscape browser and so did not produce or receive HTTP traffic characteristic of the native Windows OS browser, Internet Explorer. The last type of traffic resulted from a human actor performing various administrative tasks such as running statistics generating scripts and posting the results in Excel spreadsheet form to a web site.

There were other types of Windows NT traffic, but this traffic resulted from attacks being carried out against the Windows NT host and are not considered part of the general background traffic.

2.3 Lincoln Adaptable Realtime Information Assurance Testbed

LARIAT's two design goals are to create a system capable of real-time evaluations and to create a deployable, configurable, easy-to-use testbed that could be easily integrated into an organization's existing network. By integrating LARIAT into an existing network, IA technologies, as well as any other network component, can be evaluated in a relevant network configuration with relevant background traffic. It is also able to generate both single-component attacks and multi-component attacks, and automate many of the time consuming tasks of past IDS evaluations.

In order to achieve these goals, LARIAT is designed so that a user of the system can interact with a central director, the experiment configuration and automation tool. The director alleviates the need for the user to have detailed knowledge of the testbed itself. The director is responsible for the actual configuration and subsequent running of an evaluation. A possible LARIAT topology is depicted in Figure 1.

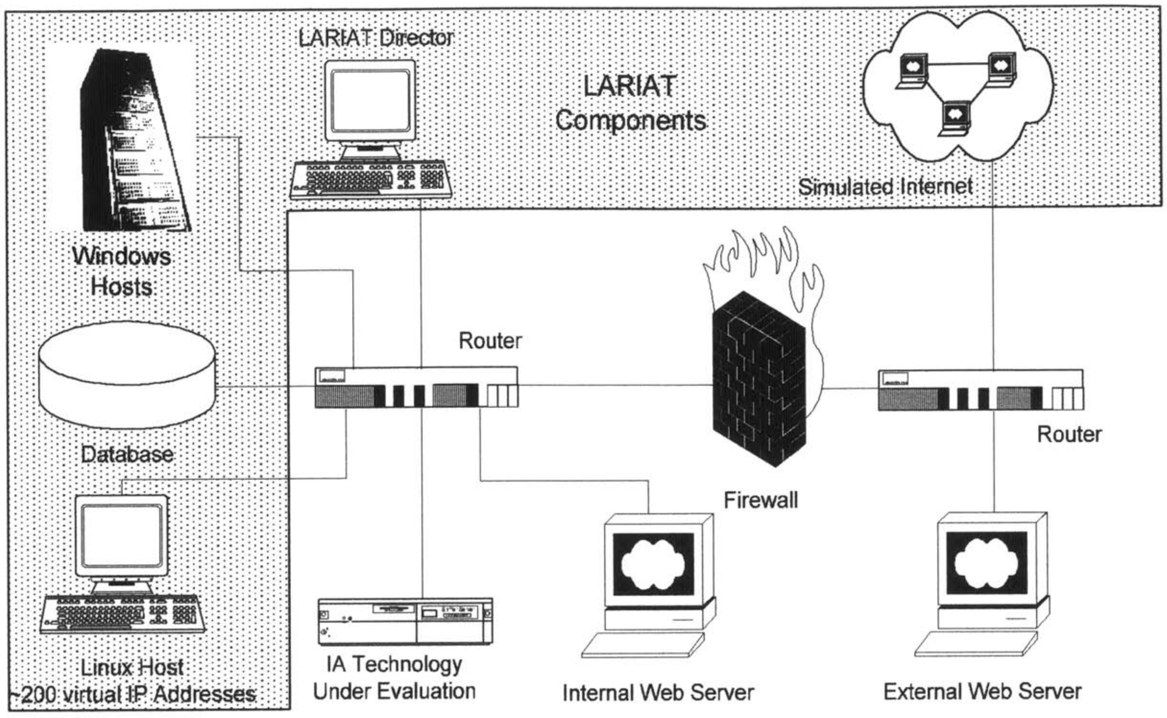


Figure 1: Example LARIAT Topology

Since WinNTGen is integrated into LARIAT, the following sections provide an explanation of the roles of different components in a LARIAT setup and a summary of the evaluation process. A more detailed description is available in the original LARIAT paper[7].

2.3.1. LARIAT Director

The LARIAT director controls almost all aspects of the testbed. A user of LARIAT needs only interact with the director in order to run experiments once the testbed is set up. The director is capable of producing network traffic profiles based on the parameters provided by experimenter. These profiles dictate the production of network traffic by the traffic generating hosts. The profiles currently in use in LARIAT were created from analyzing traffic from a United States Air Force base. In order to simplify the setup of experiments, the LARIAT director has prefabricated traffic profiles, user profiles, and attack configurations that can be modified and tailored to a particular experiment. It is not necessary to specify all aspects of an experiment every time it is run.

2.3.2. Database

The database is responsible for storing a representation of the network, the experiment parameters, the traffic and user profiles, run time logging, and experiment results. Using a database centralizes the information, eases deployment and reconfiguration of the testbed, and standardizes the structure of information.

2.3.3. Traffic Generators

A traffic generator may contain on the order of 2500 virtual IP addresses or may contain a few or even just one. It may be responsible for background traffic such as web surfing and sending emails or it may be involved solely in attacking another host or being attacked.

There is a common required functionality in the traffic generators, however. Each traffic generator must be capable of communicating with the director and the database. From the director it obtains the address of the database and directions of when to start, stop, reread parameters from the database, and report run information to the database.

2.3.4. Evaluation Overview

An experiment in LARIAT is comprised of eight steps, as depicted in Figure 2. In the first step the user selects and edits attack and background traffic profiles for the session. The remaining seven steps are automated and controlled by the director. In these steps, the director initializes the network, distributes the configuration to the traffic generators, verifies pre-conditions, runs the traffic, verifies that the traffic ran as desired and collects results, and then cleans up in preparation for a fresh run. The following is a detailed description of these steps.

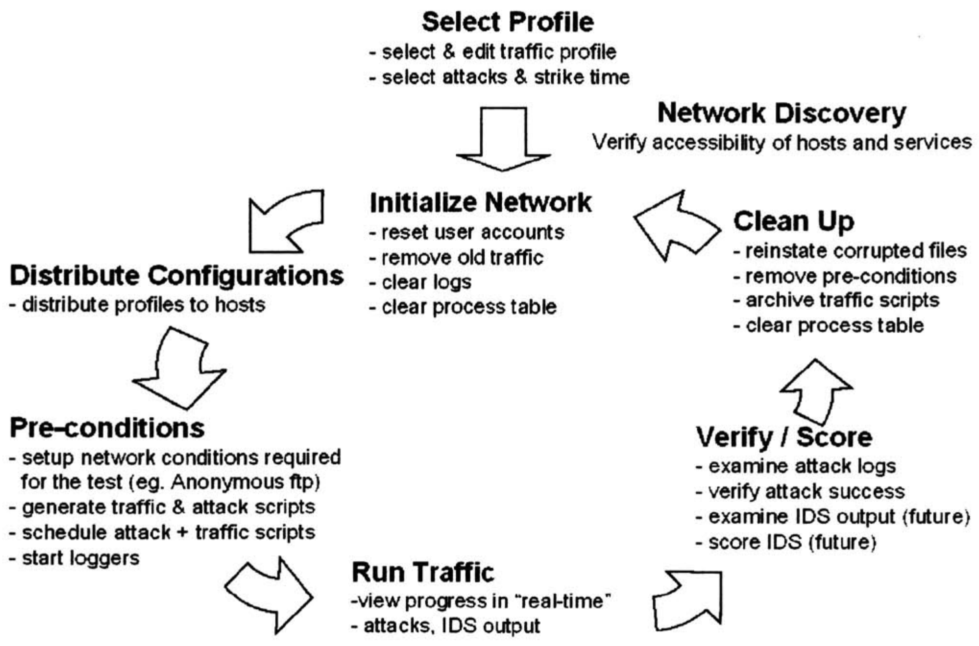

Select Profile
- select & edit traffic profile
- select attacks & strike time
Network Discovery
Verify accessibility of hosts and services
Initialize Network
- reset user accounts
- remove old traffic
- clear logs
- clear process table
Clean Up
- reinstate corrupted files
- remove pre-conditions
- archive traffic scripts
- clear process table
Distribute Configurations
- distribute profiles to hosts
Pre-conditions
- setup network conditions required for the test (eg. Anonymous ftp)
- generate traffic & attack scripts
- schedule attack + traffic scripts
- start loggers
Verify / Score
- examine attack logs
- verify attack success
- examine IDS output (future)
- score IDS (future)
Run Traffic
-view progress in "real-time"
- attacks, IDS output

Figure 2: Experiment Flow

The experimenter first selects a profile on the director for the run. As mentioned before, network traffic profiles consist of host, user, and attack configurations. The host configurations contain host details such as services offered, accounts, etc. User configurations contain information such as the type of user (secretary, administrator), frequency of ftp and telnet sessions, and periods of activity. The attack configuration contains attack components organized into scenarios that recreate the attacker actions as he performs all the steps required to penetrate a network. Several representative attack scenarios are provided[8].

Following the specification of the profile by the user, the director distributes the configuration information to a database and then indicates the start of an experiment to the traffic generators via a simple custom network protocol. The traffic generators respond by fetching the experiment parameters relevant to them from the database.

Once the traffic generators have obtained their configurations, they prepare themselves for the experiment which typically involves initializing user models and scripts that will define the behavior of the traffic generator for the duration of the experiment.

Following the preparation, the evaluation commences. Each traffic generator produces traffic according to the aforementioned scripts and models. The director does not need to be active during this phase. Real time statistics on the run can be observed, but the experimenter must be aware that the affects of this control traffic may interfere with the evaluation.

After the evaluation run or at certain points during the run, the director can query the logs on the various machines and the database to determine the success or failure of any of the scripts or items in a queue. For example, in an evaluation involving a multi-stage attack on a victim host, the success or failure of each stage may be observed from the director.

2.4 The Need for Windows Traffic

Both the DARPA IDS evaluations and LARIAT systems had network traffic produced by Windows NT computers, but lacked the capability to produce much of the traffic that is characteristic of Windows NT. As the Department of Defense and many commercial companies use more Widows-based products, IDS evaluation systems will need the capability to incorporate realistic Windows network traffic into their experiments.

Statistics taken from browser traffic in spring 2002 [14-16] indicate that Microsoft holds roughly 80 to 90 percent of the workstation OS market with Windows 2000 and XP making up between 20 to 30 percent of the total. A March 2000 study of the web-server OS market [17] indicates that Microsoft's Internet Information Server (IIS) comprises around 23 percent of all web servers, and 30 to 40 percent of commercial web servers.

These numbers indicate that there is a significant presence of both workstations and servers with some version of the Windows OS installed. The ability to produce both Windows background and attack network traffic will enable evaluation systems to perform more comprehensive assessments of IDSs.

2.5 Available Windows Traffic Production Systems

Prior to the creation of the Windows traffic generator, a survey of commercially available traffic production systems was performed to determine if they were suitable for use in an evaluation network such as LARIAT. In order for a system to be usable in an IA technology evaluation, it needs to be capable of producing a broad range of reasonably realistic network traffic whose mix and rate can be configured. Several systems were reviewed that were capable of generating Windows NT traffic in some capacity. These systems fell into three different categories.

The first category of tools is the stress testers [18-20]. These tools are used in the evaluation of an individual host's ability to handle high throughput of a specific protocol. Neither inter-packet timing nor packet contents matches what is commonly seen on real networks, so these tools are not appropriate for general use in LARIAT.

The second category of tools is used for evaluating network configuration [21]. These tools observe network traffic and provide statistics, graphs, and/or topologies of the network. The network traffic generation ability of these tools is limited and is provided only for playing back captured or generated packets in order to verify the functionality of network resources.

Finally, the third category includes tool with the combined functionality of the above tools [22].

Since all of these systems did not meet the requirements described above, it was necessary to develop a Windows NT traffic generation system.

Chapter 3

Windows Traffic Generation

3.1 WinNTGen Goals

The goal of the WinNTGen traffic generation project was to create a system capable of producing realistic Windows NT-specific traffic that would complement the existing network traffic producing capabilities of LARIAT. Specifically, the mechanisms used should allow for the generation of any type of network traffic producible by Windows NT and be extensible enough so that unforeseen sources of network traffic, such as new applications, services, etc., could be added later and easily integrated into the system.

Since most network traffic is due to humans directly or indirectly controlling applications, the WinNTGen system design includes a model of human-computer interaction (HCI). Some parts of the original DARPA IDS Evaluation included user sessions for the Linux traffic generators in which simulated users would telnet to a host, perform sets of actions such as using ftp, etc, and then log out. Similarly, traffic produced in the WinNTGen system comes as a direct result of the simulation of user actions rather than the emulation of lower level protocols. However, the models of the users' sessions in the DARPA Evaluation came directly from recorded network sessions, whereas the models in the WinNTGen system come from recorded user interactions with various applications' GUIs on the host. For the evaluation of host-based IA technologies, it is important that users are being modeled at this level since the series of events a user performs may be used to detect illicit users or suspicious behavior. For the network-based IA technologies, it makes little difference at which level the traffic generation occurs since it has no access to events occurring on the host. However, by controlling applications directly, the network traffic timings and eccentricities of application implementations are accurately reproduced. The application and operating system correctly formats, transmits, receives, and interprets the network packets.

Table 1 summarizes the differences in traffic generation models between WinNTGen, DARPA IDS Evaluations, and LARIAT.

	DARPA IDS Eval (1999-2002)	**LARIAT (2000 - Present)**	**WinNTGen (2002)**
Network traffic produced	Primarily UNIX with some Windows NT	Primarily UNIX and some Windows NT	Only Windows 2000 and above
Traffic generated by	Simulated user sessions per individual applications. Protocol level interaction.	Simulated user sessions per individual applications. Protocol level interaction.	Simulated session of user interacting with multiple applications simultaneously
Traffic Generation Model based on	Recorded network traffic	Recorded network traffic	Recorded user actions
Per physical machine capabilities	Thousands of users	Thousands of users	Single user
Network Configuration	Fixed	Variable	Variable

Table 1: Comparison of WinNTGen to LARIAT, DARPA IDS Eval

Originally, the system was to produce realistic multi-sourced multi-user network traffic. To achieve this, each physical Windows host was to log on multiple simulated users simultaneously and modify outgoing source specific host information in network packets. However due to operating system constraints discussed in later sections, the original project goals were altered. The modified project goal is to have one simulated user per physical host.

3.2 Generating the Network Traffic

In WinNTGen, a user simulator creates network traffic by interacting with multiple applications such as using Outlook, telnet, ftp, or the Windows Explorer shell. All generation of user activity occurs at the session and application layers [23]. At a session level, the user simulator models a user interacting with a set of applications simultaneously, interleaving different activities much like a real user would. At the application level, the user simulator models the way in which a user interacts with individual applications.

There are several advantages to this approach as compared to modeling lower level application or protocol behavior. This level of abstraction increases the forward compatibility of WinNTGen. The system is more resistant to problems created by upgrades in software because it only needs to be concerned with interacting with the software at a higher level, rather than having an understanding of the internal workings and protocols that produce network traffic. Also, by controlling the application directly rather than implementing the protocols it uses, timing characteristics, implementation eccentricities, and other identifying features of that particular application appear in the network traffic automatically. For example, T. Lau and E. Horvitz used a large set of distributions in modeling how web requests arrive at a server [24]. Given the existence of embedded references in HTML documents such as images, part of their modeling had to incorporate the behavior of the browser when retrieving these references. In WinNTGen, by modeling the way in which the user interacts with the browser, all of the network traffic characteristics resulting directly from the browser are already present. In Figure 3, WinNTGen can be seen controlling several different applications which use an overlapping set of protocols. The protocols in turn utilize the drivers which then use the hardware. Each of these layers can affect the timing and content of data flowing out of the system. One action at the User Model level can cause multiple packets being sent out on the network. For example, when a user accesses a shortcut for the first time that points to a Windows network share there are multiple network conversations that occur. First, the remote host name would need to be resolved via DNS or WINS. Next, authorization would be carried out using LANMAN or Kerberos authentication schemes. Finally the contents of the directory would be transferred via SMB over

NetBIOS and TCP/IP. A single user action results in multiple actions being carried out at the lower levels.

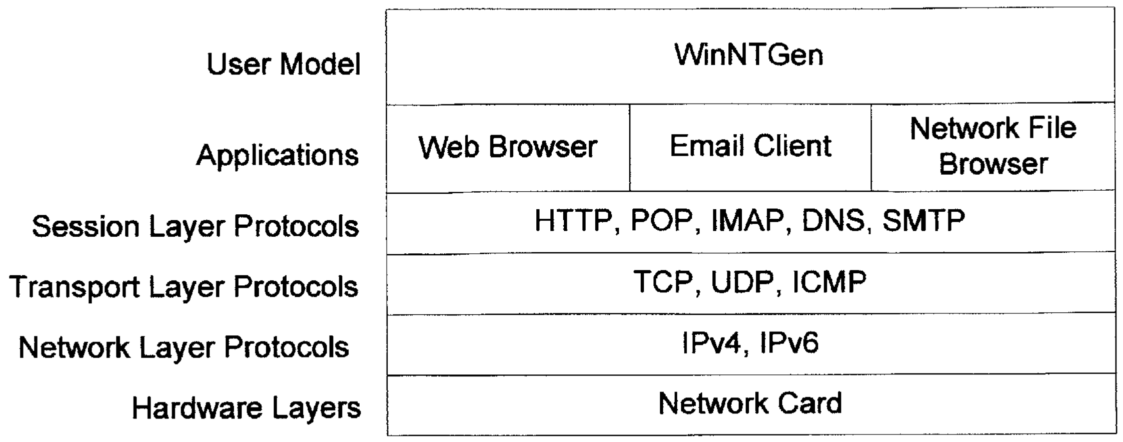


Figure 3: WinNTGen Layered Over Applications

The approach of modeling users in terms of user sessions is different from the rest of the traffic producing mechanisms currently in use in LARIAT

The Linux based network traffic production systems in use in LARIAT currently produce sets of Expect-based scripts [25] for each user session or service for the entire evaluation run. Since all of the sessions are generated prior to the start of an experiment, the traffic generators are not able to react to dynamic experiment conditions such as failures of network components. In contrast, the Windows traffic generators are producing completely real-time sessions and are able to react to changes in the testbed. For example, a simulated user might react to a downed telnet server by choosing a different one to log-in to, rather than repeatedly trying to connect the broken one.

In general, simulating user events is best done at the level at which users actually interact with the system. In the case of UNIX based systems, a sequence of user events can be represented by a character stream, the series of commands a user types. However, in the case of Windows OSs, the sequence of user events is the set interactions with the GUI. As a result, a different approach must be taken to model a user.

3.3 User Modeling

Individual users are modeled as a collection of application use state machines (AUSMs). Each AUSM encapsulates a user's behavioral pattern for a particular class of network traffic presence such as browsing the web, sending email, etc., as depicted in Figure 4. Details of the individual state machines appear in Section 3.6.

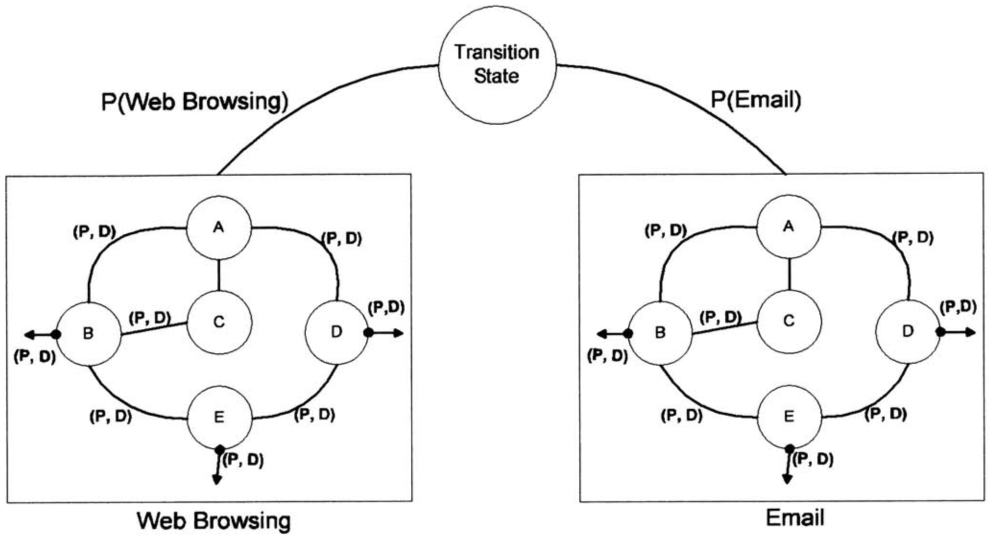


Figure 4: Collection of AUSMs

Each state within a particular AUSM is a single action that the user can do such as composing an email, sending an email, or navigating a web page as represented by A, B, C, D, and E in Figure 4. Note that the granularity of a single user action is dependent upon the AUSM.

The collection of AUSMs will be referred to as the user simulator and is itself a state machine consisting of a central transition state and all of the AUSMs in use. Note that many states within an AUSM have edges that transition out of the AUSM. This simulates the way in which real users interleave tasks when using a computer. For example, a user may be surfing the web and then decide to email a link he ran across to a friend. Upon returning to surfing the web, the user will start exactly where he left off. The transitions to the different states and different AUSMs form a serial stream of user actions. There is currently no standard mechanism through which AUSMs can interact with one another. As a result, more complicated series of events are not directly supported, such as a simulated user copying a URL from a browser window and then pasting into an email to send to a friend. Although this behavior is possible to emulate, it has not been explicitly built into the system in the interest of simplicity.

A more formal description of the user simulation process is as follows. At the beginning of a simulation session, the user simulator selects an AUSM at random according to the AUSM-use probabilities, denoted as P(Email) and P(Web Browsing) in Figure 4. Inside the chosen AUSM, a start state is entered such as "Open a web browser." Then, transitions occur between the different states depending on the edges. The edges have two values associated with them: *P*, the normalized probability that this edge is taken out the current state versus another, and, *D*, the set of parameters for some distribution that dictates the duration until this transition occurs. Transitions between states inside the module are made by the algorithm appearing in Figure 5.

```
Transition(State CurrentState)
Begin
      Edge NewEdge = CurrentState.PickEdge();
      Wait(NewEdge.GetValueFromDist());
      CurrentState = NewEdge.DestinationState;
End

State::PickEdge()
Begin
      Total = 0;
      Value = Rand();
      For(Edge = FirstEdge; Edge; Edge.GetNextEdge())
            Total = Total + Edge.P;
            If(Value < Total) Return Edge;
      End
End

Edge::GetValueFromDist()
Begin
      Return GenerateValueFromDist(Edge.D);
End
```

Figure 5: State Transition Algorithm

From the current state, pick an edge at random out of the state according to the weights and a time to wait until transition according to the distribution for that edge. Finally after waiting that amount of time, make the transition. This process is repeated.

In order to vary the behavior of the AUSMs and, more generally, the simulated users, the concept of a user class is introduced. Each class of user is defined by a different profile. Examples of classes include secretaries and software developers. Each user class is defined in terms of a profile. A profile for a particular user class is a collection of parameters that define properties of the simulated user's behavior such as the frequency of email sent and file types of attachments. For example, a user model of type secretary might have high transition probabilities to the email AUSM, and the internal parameters for the email AUSM might include the email addresses of the group for which he works and maybe even some typical emails that the secretary would send.

This approach of completely defining the behavior for individual users is similar to that of the work in the DARPA IDS Evaluation [4]. Although in that research, the user models were monolithic. For example, the traffic generator would have the simulated user telnet into a server, use finger and ping among other utilities, and then log out the user. The telnet traffic generator had to be aware of the existence of the use of the other utilities.

The traffic generators in LARIAT are similar to the traffic generators from the DARPA IDS evaluation in that a single user is chosen to log in via telnet to perform some action. However in LARIAT, the simulated user performs actions that produce mainly one type of traffic. This allows a more precise, higher level control of the distribution of protocols seen during an experiment. For example, instead of logging into a server via telnet, using ftp, using ping, using nslookup, and then logging out as in the DARPA IDS evaluations, the simulated user would log in, use ftp for some period of time, and then log out.

The AUSMs currently in use in WinNTGen do not model user behavior in terms of goals, operators, methods, and selection rules (GOMS)[26, 27]. In that research, a user is modeled by first selecting a set of goals that user wants to accomplish, and then deciding upon the series of atomic actions the user should take in order to reach that goal. The AUSMs currently in use in WinNTGen provide a set of actions the user can take at any point with no notion of a greater intent. The knowledge that is embedded into the AUSM allows it to reproduce typical courses of actions that the user takes, such as opening an email client, then checking email, sending email, and finally closing email client, thereby creating a user plan[28]. If a greater level of sophistication is desired in the user models in the future, it can be easily added by updating only the AUSM and leaving the rest of the system intact.

In order to model user behavior on a larger time scale, such as over the course of a day or work week, several mechanisms are employed. For controlling when a user from a particular class logs in for the day, a mean daily start, mean daily finish, and distribution parameters for both are used. This allows an experimenter to define when simulated users typically start using network resources and when they stop. For example a group leader user class may have a mean daily start of 8:30am with a low variance distribution indicating that he usually arrives the same time every day whereas a developer may arrive uniformly between 9:00am and 12:00pm.

Also, during the course of the day user activity rates change. For example, during lunch there is a noticeable drop in network activity. In order to model this, hourly rates are used. The hourly rates simply scale the level of user activity by modifying inter-event and inter-AUSM timings.

3.4 Obtaining Behavior Data

In order to obtain realistic timings for the simulated users, it was necessary to create a system for recording real user's timings. A component object model (COM) plug-in [29] was created capable of recording user events from Windows Explorer and Internet Explorer [30]. Events were recorded from these applications in order to determine the user behavior for network file share browsing and internet browsing, respectively. The plug-ins recorded user actions as a serial stream of events. Examples of an event stream can be seen in Figure 6. Note that in the figure, events are numbered according to process ID in order to differentiate the parts of the stream produced by different applications.

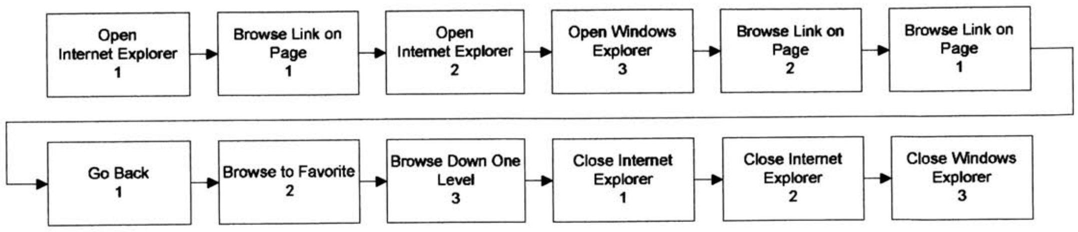


Figure 6: Event Stream

These events were then used to set the weights and transition times of the AUSM as discussed in the next section. The actual event streams that were used to train the models discussed in this paper came as a result of seven members of the Information Assurance group performing normal daily internet and file browsing activities over the course of one week.

Note that the daily log in and log off times were not recorded and modeled and were chosen arbitrarily with the work day defined to be 8:30am to 5:00pm with an exponential distribution that dictates the deviance from the mean for both. A more accurate model of login and logoff times will be developed in the future.

3.5 Defining Application Use State Machine Behavior

There are two possible approaches that can be used to define AUSM behavior. The first approach is to train the AUSMs from the stream of recorded user events. This approach has the advantage of realism, but requires real user data in order to set the transition parameters in the AUSM. The Windows Explorer and Internet Explorer AUSMs' parameters were set using this approach.

The other approach is to explicitly define the frequency of a certain states, such as *Send Email,* and then use these frequencies to set the transition parameters in the model. The advantage of this approach is that it allows an experimenter to set rates of certain actions and does not require the AUSMs to be trained. The produced traffic will not be realistic in terms of user timings, but can be used to stress components of IA systems. The parameters in the Outlook AUSM were set using this approach.

3.5.1. Training AUSMs from Event Streams

In order to define the behavior of an AUSM based on a stream of recorded events, each recorded stream was fed through a program which set the parameters of state machines that matched those found in the various AUSMs. As depicted in Figure 7, the program separated each stream according to event source (i.e. the application that generated it) and process ID. Each of these smaller streams was then fed through the appropriate internal state machine.

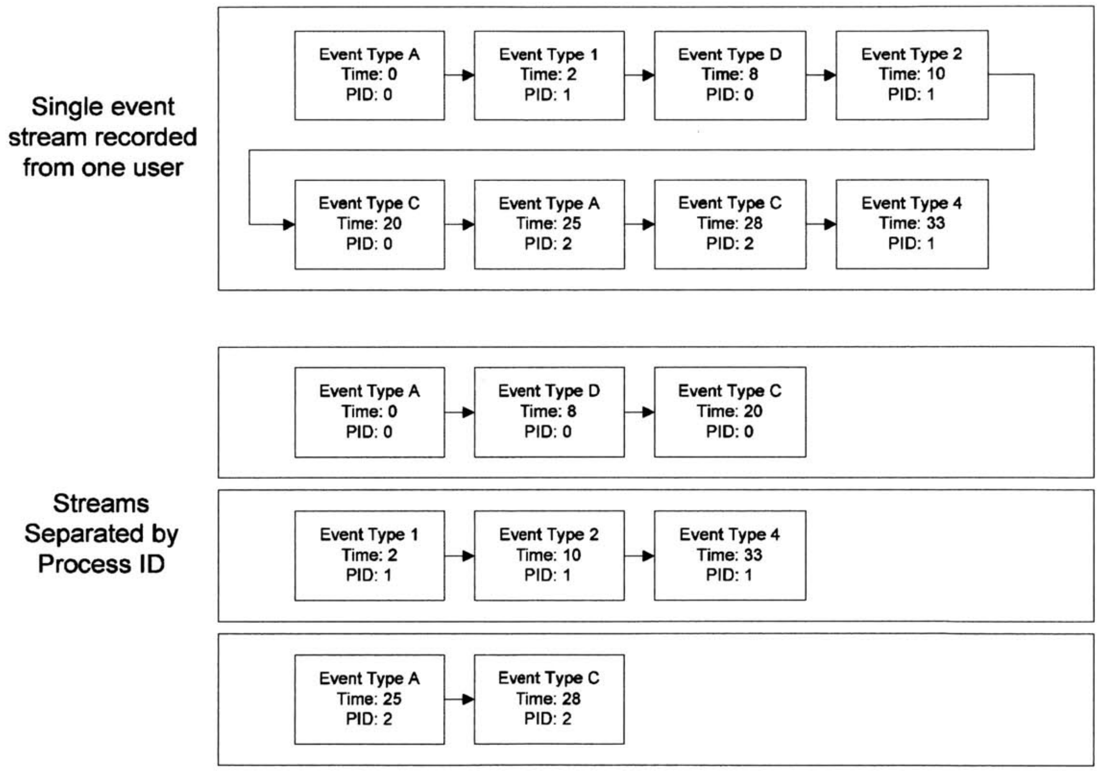

Single event stream recorded from one user
Event Type A Time: 0 PID: 0
Event Type 1 Time: 2 PID: 1
Event Type D Time: 8 PID: 0
Event Type 2 Time: 10 PID: 1
Event Type C Time: 20 PID: 0
Event Type A Time: 25 PID: 2
Event Type C Time: 28 PID: 2
Event Type 4 Time: 33 PID: 1
Streams Separated by Process ID
Event Type A Time: 0 PID: 0
Event Type D Time: 8 PID: 0
Event Type C Time: 20 PID: 0
Event Type 1 Time: 2 PID: 1
Event Type 2 Time: 10 PID: 1
Event Type 4 Time: 33 PID: 1
Event Type A Time: 25 PID: 2
Event Type C Time: 28 PID: 2

Figure 7: Event Stream Separation

Each of the internal state machines has a predefined start state, such as *Open Internet Explorer* that it assumes will appear first in each stream. Once the start event appears in the stream, the program traverses the state machine according to the events in the stream as depicted in Figure 8. The application keeps track of how often a particular edge was followed out of a state as well as the time until that transition occurred. Once all the separated event streams had been fed through their respective state machines, the transition probabilities and transition time distributions were determined.

Note that it is not necessary to have a static model of the user's actions (i.e. one with a fixed number of states). Only for simplicity were the static models used. Since the internal behavior of the AUSMs is not predefined by WinNTGen, it is completely possible to use any type of model of user behavior such as simply playing back recorded user events.

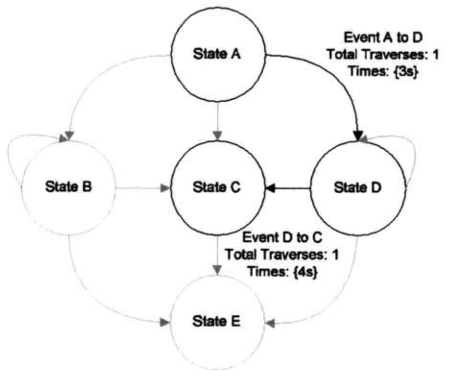


Figure 8: Traversing the State Machine

A transition probability was assigned to each edge leading out of a particular state by normalizing the number of times each edge was taken out of that particular state. Only the current state rather than the series of states leading up a transition was considered when determining transition probabilities, or more formally, the probability model was first-order. Empirical evidence from research conducted by L. Jung Jin and R. McCartney [31] suggests that the two approaches produce similar probabilities when considering sequences of user actions.

In modeling the transition time for each edge in the various models, a single distribution type was chosen rather than having a separate one for each edge. This was done due to the current simplicity of the models and the lack of extensive user data. A more complicated distribution was deemed unnecessary since a cursory inspection of the collected user data indicated that the distribution shapes were sufficiently similar.

This general distribution was derived from data collected from Internet Explorer usage since the vast majority of recorded user data came from there. For each edge in the Internet Explorer model, a CDF of the transition times was plotted in order to determine the features of the distribution. An example of the general shape of the CDF of the collected data can be seen in Figure 9. This particular plot is of the elapsed time between a user following a link on a web page and then following a link on the new page.

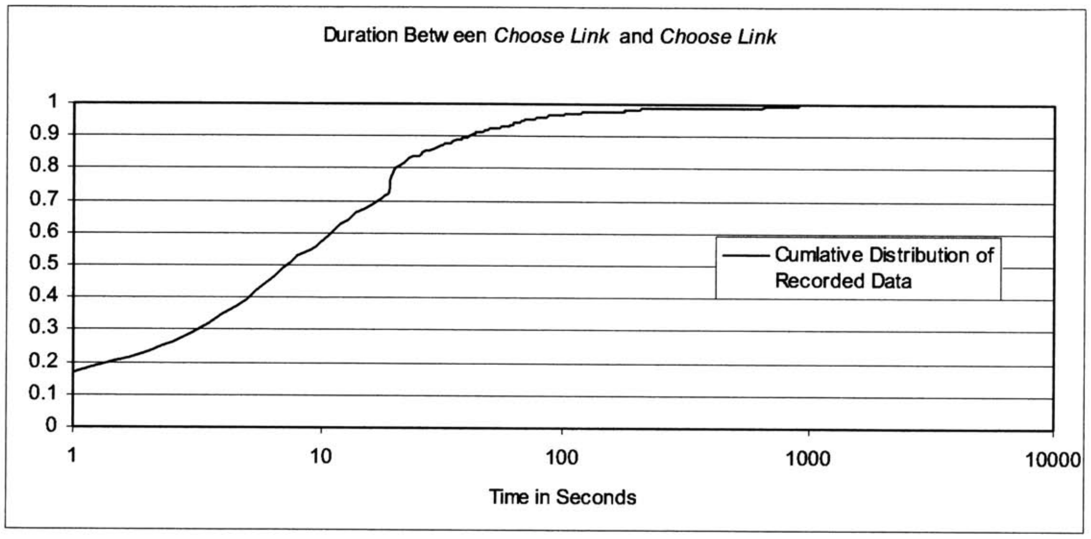


Figure 9: CDF of Transition Durations

An appropriate distribution would need to capture two features of cumulative distribution of the recorded data. The first feature is the large number of data points around zero indicating that a user performed an action then very quickly performed another action. The second feature is the heavy tail that is present indicating that sometimes users perform actions and then do nothing for long periods of time as previously noted by V. Paxon and S. Floyd [32].

The first feature can be captured by a general exponential distribution (Equation 1) since it can be heavily weighted around zero. However, the exponential distribution does not have heavy tails and therefore does not capture the second feature. The Pareto distribution (Equation 2) does have heavy tails, but it is not heavily weighted around zero. Therefore, in order to model the observed data it was decided that a hybrid distribution would be used. The hybrid distribution uses an exponential distribution to produce values close to zero and the Pareto distribution to produce values for the tail as seen in Equation 3.

$$D(x) = 1 - e^{-\lambda x} \qquad \text{Equation 1}$$

$$D(x) = 1 - \left(\frac{\beta}{x}\right)^{\alpha} \qquad \text{Equation 2}$$

$$D(x) = \begin{cases} 1 - e^{-\lambda x} & x \le b \\ 1 - \left(\frac{\beta}{x}\right)^{\alpha} & x > b \end{cases} \qquad \text{Equation 3}$$

The hybrid distribution has a breakpoint parameter, b, in addition to the parameters for the Pareto and the exponential distributions. The breakpoint is the point at which the values change from appearing to be exponentially distributed to being distributed according to a Pareto and is derived from a ratio of the low mean values to the high mean values. To derive the breakpoint, the values are bucketed according to the ratio, the top n% in one bucket and the remaining 100-n% in the other bucket. The breakpoint is then chosen to be greater than the maximum value of the lower bucket and lesser than the minimum value in the higher bucket.

The hybrid distribution fitted to the recorded data appearing in Figure 9 can be seen in Figure 10. For this plot, the values used for λ, α, β, and b are 9, 1.1, 5, and 10 respectively. The breakpoint in this case was based on inspecting and fitting this particular data.

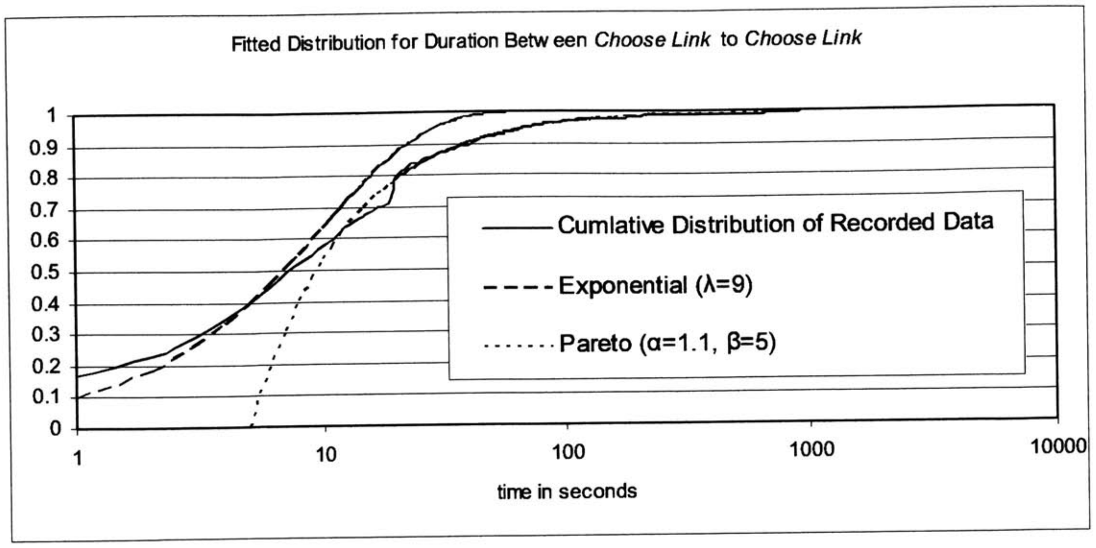


Figure 10: Fitted Hybrid CDF

In order to set the distribution parameters for the other edges, a general approach was used so that each distribution in the future would not need to be explicitly fit by hand. After examining the transition-time data for all edges, it was apparent that roughly 40% of the transition times were produced by a heavy tailed process and 60% by a light tailed process. Therefore, the breakpoint for each edge was set so that 40% of the data lie above the breakpoint and 60% of the data lie below.

In order to derive the actual distribution parameters, the inverse CDF for the Pareto (Equation 4) and general exponential (Equation 5) distributions were solved for their parameters and evaluated at x, where x is equal to the breakpoint. For the Pareto distribution, α was set to be 1.1 since there was not enough high value data to justify deriving a different value for each transition.

$$\beta = \frac{x}{e^{\frac{-\ln(0.6)}{1.1}}} \approx \frac{x}{1.59} \qquad \text{Equation 4}$$

$$\lambda = \frac{-x}{\ln(0.4)} \approx \frac{x}{0.91} \qquad \text{Equation 5}$$

3.5.2. Setting AUSM Parameters from State Frequencies

The other approach to define the behavior of the model is to explicitly set how often each state should appear in the produced event stream. This allows a researcher, for example, to set roughly how much POP traffic he would like to see in the generated network data in relation to SMTP traffic while still maintaining the constraints present in the model such as the need for a user to log in to a POP server before he can check his email.

Since there is more concern for rates than realistic state transitions, a simpler transition model can be adopted. The motivation behind adopting a simpler model is to reduce the complexity of the calculations inherent in working backwards from state frequencies to edge probabilities and distribution parameters.

For this paper, the discrete time Markov model is used. In using a discrete time model, the simplification is made that every edge takes a fixed amount of time thus leaving only the edge probabilities to be determined.

The problem is as follows. Given $r_1, r_2, ..., r_k$, the rate of occurrence of each of the states, p_{ij}, the probability of transitioning from state *i* to state *j* for all *i* and *j*, needs to be determined. First the rates $r_1, r_2, ..., r_k$ need to be converted into $\pi_1, \pi_2, ..., \pi_k$, the probabilities that the current state in a model will be 1, 2,...,k at some time in the future. To do this, each rate *r* needs to be normalized by the total of all rates (Equation 6).

$$\pi_i = \frac{r_i}{\sum_{j=1}^{k} r_j} \qquad \text{Equation 6}$$

This also guarantees the constraint present on Markov chains (Equation 7) is satisfied.

$$\sum_{j=1}^{k} \pi_j = 1 \qquad \text{Equation 7}$$

Next, the balance equations (Equation 8) are set up in order to determine the probabilities.

$$\pi_i = \sum_{j=1}^{k} \pi_j p_{ji} \qquad \text{Equation 8}$$

Now, values for p_{ji} can be chosen with the only other constraint being that all the probabilities leaving a state must sum to one (Equation 9). Note there can be multiple solutions for a given set of rates.

$$\forall i, 1 \le i \le k, \sum_{j=1}^{k} p_{ij} = 1$$ Equation 9

Once the edge probabilities have been chosen, the edge transition time for every edge is set (Equation 10). In order to clarify the equation with an example, if there is a total rate of ten events per minute then transition the time for each edge should be six seconds.

$$\Delta t = \left(\sum\nolimits_{j=1}^{k} r_j\right)^{-1}$$ Equation 10

3.6 Implemented Application Use State Machines

This section presents the Internet Explorer and Windows Explorer AUSMs, which were trained by recorded user data, and Outlook model which was defined in terms of state frequencies.

3.6.1. Internet Explorer

The Internet Explorer AUSM models a user's interaction with the web browser as depicted in Figure 11 with transition parameters appearing in Table 2. Note that the transition probabilities also appear in the figure with the value of P(A→B) appearing next to B. While this state machine could be used for modeling a user interacting with any web browser, Internet Explorer was chosen since it is predominantly used on the Windows OS.

The state machine attempts to capture the manner in which users browse the web by defining an entrance and exit state (e.g. open and close the browser) and then a group of completely connected intermediate states that define how a user browses web pages (e.g. follow a link on the current page, browse to a favorite or book-marked link). This is a simple model and does not capture more complicated behaviors such as how link appearance affects its probability of being chosen, which favorites a user chooses most often, or how a user refines web searches [24], although this capability can be added if a more sophisticated model is needed.

Note that the *Unknown* state refers to the case when the source of the URL being navigated to is unknown such as when a user types the URL directly into the address bar, clicks on it from an email, or the URL is dynamically generated by JavaScript on the page.

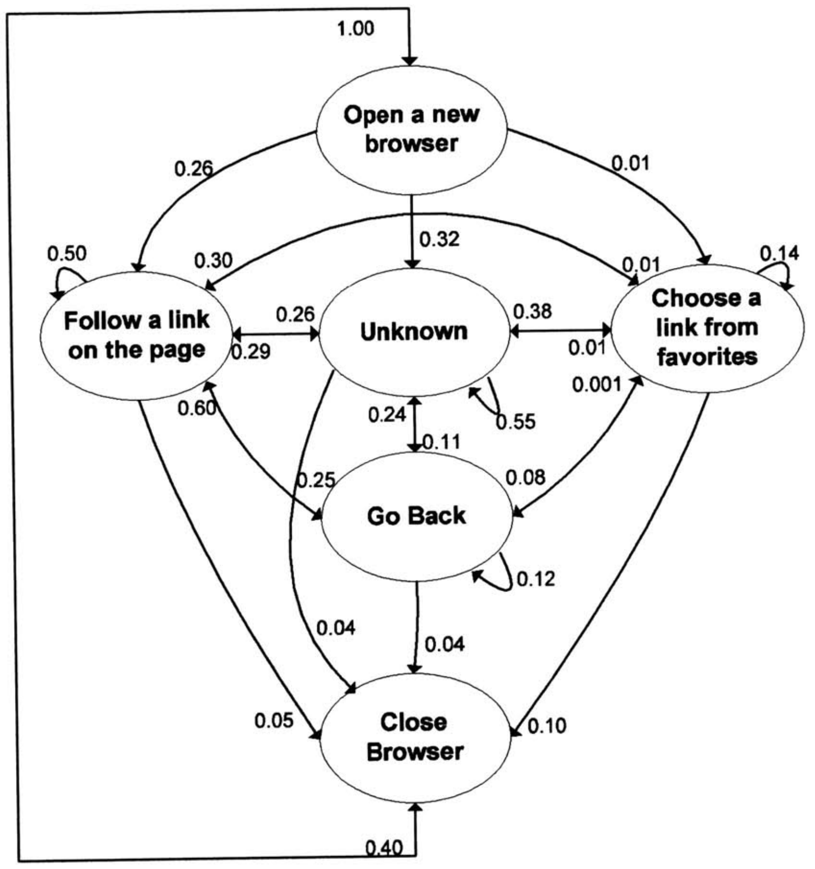


Figure 11: Internet Explorer AUSM

Link		Probability	λ	α	β	b	n
Open Browser	Follow Link	0.26	4.40	1.1	7.69	7	41
	Choose Favorite	0.01	6.29	1.1	10.99	10	2
	Close Browser	0.40	6.92	1.1	12.09	11	63
	Unknown	0.32	4.40	1.1	7.69	7	50
Follow Link	Follow Link	0.50	6.29	1.1	10.99	10	1037
	Go Back	0.25	9.43	1.1	16.48	15	519
	Choose Favorite	0.01	6.29	1.1	10.99	10	21
	Unknown	0.20	6.29	1.1	10.99	10	415

Link		Probability	λ	α	β	b	n
	Close Browser	0.05	9.43	1.1	16.48	15	104
Choose Favorite	Follow Link	0.30	5.03	1.1	8.79	8	14
	Go Back	0.08	4.40	1.1	7.69	7	4
	Choose Favorite	0.14	9.43	1.1	16.48	15	7
	Unknown	0.38	5.66	1.1	9.89	9	19
	Close Browser	0.10	2.52	1.1	4.40	4	5
Go Back	Follow Link	0.60	6.29	1.1	10.99	10	469
	Go Back	0.12	3.14	1.1	5.49	5	94
	Choose Favorite	0.001	6.29	1.1	10.99	10	1
	Unknown	0.24	4.40	1.1	7.69	7	188
	Close Browser	0.04	3.77	1.1	6.59	6	45
Unknown	Follow Link	0.29	6.29	1.1	10.99	10	195
	Go Back	0.11	3.14	1.1	5.49	5	74
	Choose Favorite	0.01	4.40	1.1	7.69	7	7
	Unknown	0.55	6.29	1.1	10.99	10	367
	Close Browser	0.04	6.29	1.1	10.99	10	27
Close Browser	Open Browser	1	1.89	1.1	3.30	3	17

Table 2: Internet Explorer AUSM Transition Parameters

3.6.2. Windows Explorer

The Windows Explorer AUSM attempts to capture the way in which users browse a network share directory tree as depicted in Figure 12 with the transition parameters appearing in Table 3. The entrance state is a user opening up an Explorer window for a network path such as \\somemachine\someshare. The exit state is the user closing that particular browser window. The intermediate states model how a user navigates a directory tree on a remote share.

Once again this is a simple model that only captures a user's directory browsing style. It does not take into account which directories are browsed most frequently by users or other more complicated behaviors.

Also, the file browsing AUSM does not currently attempt to open any files it finds while traversing the directory tree. If this type of behavior is desired, another state could be added, *Open File*, which would simply call ShellExecute(NULL, "open", *Filename*, *CurrentDir*, SW_HIDE) where *Filename* is the file to be opened. This would let the shell choose the application that is capable of opening files for the particular type. This would appear more realistic in terms of network traffic since the application would access the file across the network in a realistic way. For example, if the file was a media stream, it would most likely be accessed in a linear fashion.

This functionality was not implemented in the current version of this AUSM since every call to ShellExecute resulted in another application being started and over the course of a long test run this would slow the machine as more memory and CPU resources were utilized. Compensating for this involves determining which applications were launched as a result of the ShellExecute and then terminating them after some amount of time. Modeling arbitrary application use was not approached in this thesis and so was not included in this AUSM.

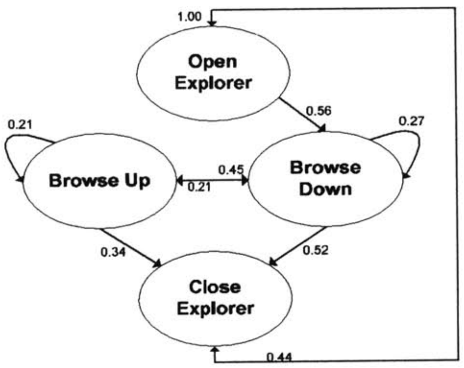


Figure 12: Windows Explorer AUSM

Link		Probability	λ	α	β	b	n
Open Explorer	Browse Down	0.56	3.77	1.1	6.59	6	22
	Close Explorer	0.44	2.51	1.1	4.39	4	17
Browse Down	Browse Down	0.27	3.14	1.1	5.49	5	19
	Browse Up	0.21	2.51	1.1	4.40	4	15
	Close Explorer	0.52	3.14	1.1	5.49	5	37
Browse Up	Browse Down	0.45	1.89	1.1	3.30	3	30
	Browse Up	0.21	1.89	1.1	3.30	3	14
	Close Explorer	0.34	1.89	1.1	3.40	3	23
Close Explorer	Open Explorer	1	2.51	1.1	4.40	4	52

Table 3: Windows Explorer Model Transition Parameters

There are several interesting points to note about the transition parameters. First, the existence of more data points around zero (i.e. lower λ parameters) as compared to the Internet

Explorer data indicates that users browse directory trees faster than the internet since directory trees have a more rigid hierarchical structure and users are generally file browsing for a specific file. Second, the fact that roughly half of the time the user does no browsing at all after opening an Explorer window would indicate that users have shortcuts to network shares they use most.

3.6.3. Outlook

The Outlook AUSM controls Microsoft Outlook. Currently the model, as depicted in Figure 13, only encapsulates the behavior surrounding sending and receiving emails, but it would be straightforward to add such functionalities as adding a contact to the address book or scheduling an appointment.

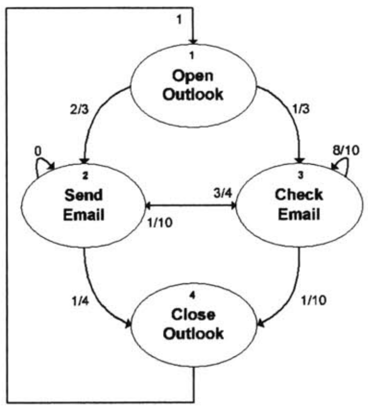


Figure 13: Outlook AUSM

Instead of defining the Outlook model edge parameters according to recorded user data, they will be derived from the rates of occurrence using the approach described in Section 3.5.2.

First the balance equations for the AUSM are derived (Equation 11).

$$\begin{aligned}\pi_1 &= \pi_4 p_{41}\\ \pi_2 &= \pi_1 p_{12} + \pi_2 p_{22} + \pi_3 p_{32}\\ \pi_3 &= \pi_1 p_{13} + \pi_2 p_{23} + \pi_3 p_{33}\\ \pi_4 &= \pi_2 p_{24} + \pi_3 p_{34}\end{aligned} \qquad \text{Equation 11}$$

Next, the state frequency rates for the experiment are set. Assume that the experimenter would like to see the simulated user send emails at a rate of four times per minute and check emails at a rate of twenty times per minute. The number of times the simulated user opens and closes Outlook is of little importance to the experimenter and these will be set arbitrarily at three times per minute each. The two rates must be equal since every time the simulated user enters the state *Close Outlook* he must transition to the state *Open Outlook.* The steady state probabilities for *Open Outlook*(π_1), *Check Email*(π_2), *Send Email*(π_3), and *Close Outlook*(π_4) are therefore 3/30, 20/30, 4/30, and 3/30 respectively. Substituting in the steady state probabilities and solving with the aforementioned constraints yields the possible solution below (Equation 12).

$$\begin{aligned} &p_{12} = \tfrac{2}{3}, p_{13} = \tfrac{1}{3} \\ &p_{32} = \tfrac{1}{10}, p_{33} = \tfrac{8}{10}, p_{34} = \tfrac{1}{10} \\ &p_{23} = \tfrac{3}{4}, p_{22} = 0, p_{24} = \tfrac{1}{4} \\ &p_{41} = 1 \end{aligned} \qquad \text{Equation 12}$$

As explained before, the transition time is set to two seconds per edge since thirty transitions per minute need to occur.

Chapter 4

Implementation

4.1 Overview

This chapter provides a detailed explanation of the implementation of the Windows Traffic generator. Section 4.2 provides a summary of the attempted implementations and what prevented those approaches from working, as well as reasoning behind the decisions leading to the current implementation. Section 4.3 details the actual implementation of WinNTGen.

4.2 Abandoned Goal: Multiple Virtual Source Addresses

Initially, the goal was to implement a Windows NT network traffic generation system capable of producing traffic from one host that would appear to be from multiple physical Windows NT hosts. This is similar to the existing LARIAT Linux network traffic generators. A single host traffic generation system capable of producing multi-sourced network traffic would be more scalable than a system in which each host could only produce network traffic that appeared to be from only that host. To achieve this, the project would have had two major parts. The first part would be the actual user simulator capable of simulating the actions of multiple users. The second part would be the mechanism through which source IP addresses and host information would be changed so as to make the actual network traffic appear to be from multiple Windows hosts.

Several solutions for single-host multi-sourced Windows network traffic were considered and subsequently rejected for multiple reasons, including unacceptable losses of realism, low per-host throughput, and lack of forward compatibility. The problems arose strictly from implementing the second major part of the system that changes the source host information.

4.2.1. User Space Filters

The first attempt at a solution was to use a Winsock 2 layered transport service provider[33]. The service provider is implemented in a dynamic link library (DLL) and sits between the Winsock 2 API functions [34] and the base service providers (e.g. TCP/IP) in user space as depicted in Figure 14. This solution involved using the existing network configuration mechanisms on Windows to define multiple IP addresses on a host. Once the multiple IP addresses were defined, the transport service provider could modify an application's calls to the *bind* and *connect* API functions for a particular socket before they reached the base service provider. Upon intercepting one of these calls, the socket would be bound to one of the locally defined IP address based on process ID of the caller.

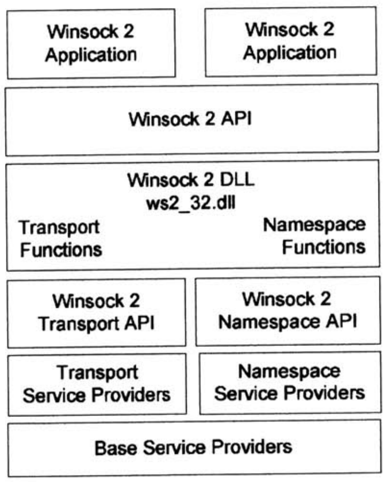


Figure 14: Layered Service Provider

There are multiple advantages of this approach. It is forward compatible since it only uses the Win32 API. It is lightweight since even though each process will load this DLL into their virtual address space, only one copy need be present in physical memory. Finally, it is capable of catching all user mode socket operations since at some level they must use the sockets API to make a connection.

The main disadvantage of this approach and the reason that it was abandoned is that none of the network traffic originating in the kernel can be filtered, such as network file sharing or active directory browsing. Drivers and services use the native Windows network stream mechanism, the transport data interface (TDI), in order to send network packets. Winsock is in fact a user mode wrapper that translates user space application calls into IO control calls for a kernel mode driver which makes the connection on the user space application's behalf. Section 4.2.2 gives a more in-depth explanation of network streams at the kernel level.

The other disadvantage to defining multiple IP addresses for a single Windows host is that since Windows is aware that it has multiple IP addresses it will communicate this fact to servers and other hosts when using certain protocols [35] such as WINS and NetBIOS, thus defeating much of the desired realism in the Windows NT specific traffic.

When performing evaluations of networked based IA components, especially more sophisticated ones, it is important that the network traffic produced by a single network traffic generating host either appear to be only from that host or appear to be from completely different hosts. Semi-realistic network traffic has the potential of producing many false alarms on the networked based IA component since the network traffic will most likely be flagged as suspicious.

Even though this solution was abandoned as the main IP traffic generation approach, this method of intercepting user space socket calls could still used by WinNTGen in the future in cases where an experiment's goals warrant more volume in the form of non-Windows specific traffic protocols such as HTTP and POP. This type of traffic can be multi-sourced easily using the described method with few artifacts linking the different packets that are generated to a single physical host.

4.2.2. Kernel Space Filters

In order to catch all network traffic regardless of whether it originated from user space or kernel space, a two layer system of filters was designed that would allow WinNTGen to modify all network packets leaving and entering the system.

Host information in packets leaving the system would be modified based on the user token of the process responsible for producing the traffic. This way, from a network standpoint, all processes owned by a user would appear to be running on a separate physical host than processes owned by a different user. Packets entering the system would need to be modified so as to not be rejected as they pass back up the driver stack to the destined application or service.

The upper layer filter would be responsible for tagging any data originating from kernel or user space bound for the network based on its original source. For example, this would include network bound data directly originating from applications such as the result of a *socket* or *send* call, or indirectly as in the case of an application calling *CreateFileEx* on a file residing on a remote device. The upper layer filter is needed since it will be intercepting calls while the context of these calls (i.e. the process making these calls) can still be determined. Once the data has reached the network card, there is no way to determine which application produced it unless it is tagged or exists in a table somewhere.

One part of the upper filter layer is a kernel driver that layers over TCPIP.sys as depicted in Figure 15. This filter driver implements the transport driver interface (TDI) as TCPIP.sys does and sits at the top of the driver stack thereby intercepting all calls originally bound for TCPIP.sys. After modifying the appropriate data, the driver uses the TDI interface of TCPIP.sys to pass on the calls. The difference between this driver and the Winsock service provider mentioned before is that this upper layer driver will catch all of the socket style requests. It is important to note that the upper layer filter would not need to modify incoming traffic since the OS would properly hand the data to the right application or service.

There would also need to be another part of the upper layer filter handles calls that indirectly create network traffic such as *CreateFileEx*. This part was never built as the approach was abandoned before this stage of implementation was reached.

The lower level filter consists of a single driver, a network driver interface specification (NDIS) miniport, which is able to modify packets right before they are sent out on the network (also depicted in Figure 15). It would need to modify the source IP address and any host specific data present in the outgoing packets according to the tagging performed by the upper layer filter. It would also need to monitor all packets on the network so as to listen for any packets bound for any of the virtual hosts being emulated by the machine. Upon receiving an applicable packet, it would need to modify the packet in such a way so that it would not be rejected as it is passed up the network driver stack.

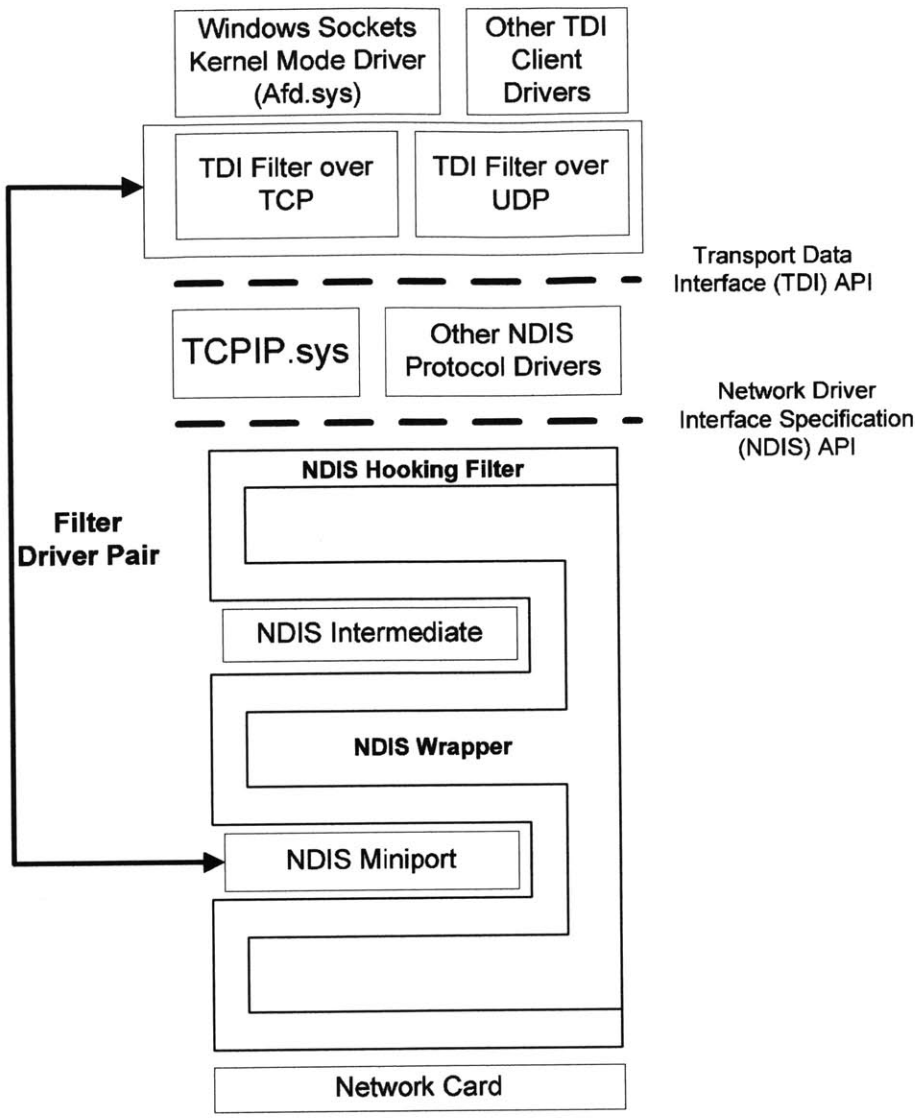


Figure 15: Cooperating Kernel Mode Filters Design

Initially, using a pair of kernel drivers as described appeared to be ideal. However, such was not the case since there is a large quantity of application layer data present in each packet that reveals that the data is originating from a single host. This data can be extremely difficult to change. For example, Kerberos traffic contains encrypted information about the hosts that are communicating [36]. When a client in a Kerberos realm transmits a ticket across the network, part of the ticket is the client's identity along with its network address encrypted in the server's private key. Changing the network address in this case is not feasible.

Other problems existed with solution as well. The development time for kernel drivers is considerably longer than a user space program of equal size and complexity due to difficulties in debugging and poorly documented events that occur in the kernel. Some of the system functions that were to be used were undocumented and therefore might not be supported in future releases of the operating system. For these reasons this approach was abandoned as a method for modifying network packets.

4.3 Final Goal: Single Source Address

The final design of WinNTGen assumes one simulated user per physical host in which only user mode applications are manipulated. This solution allows total realism and forward compatibility since all simulation happens though the control of user-space applications. The disadvantage is that the number of hosts scales with the number of simulated users, which is a large hardware cost. However, this cost may be mitigated by using a package such as VMWare [37] on a machine of sufficient resources.

4.3.1. Overview

WinNTGen consists of two parts as depicted in Figure 16. The first part is a substitute graphical identification and identification (GINA) module, herein referred to WinGenGina. This part is responsible for communicating with the LARIAT director and the database and for logging in simulated users. The second part is an application, herein referred to as WinGenApp, which is run automatically when the simulated user logs in. WinGenApp is responsible for interacting with installed applications via AUSMs according to the simulated user's profile. When the time comes for the user to log-out, WinGenApp terminates the user session and logs the user out, thus returning control to the WinGenGina. A high level flow chart for the entire traffic generation process appears in Figure 17. In the figure solid lines represent program or application flow and dotted lines represent network communication. Note that actions can occur outside of the normal experiment flow. For example, any time the LARIAT director indicates that results should be written to the database, WinGenGina will do so regardless of the current status of the experiment.

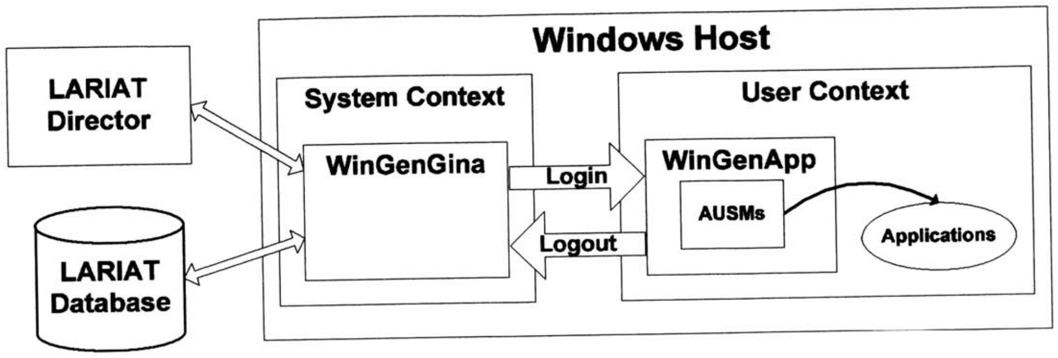


Figure 16: WinNTGen High Level Structure

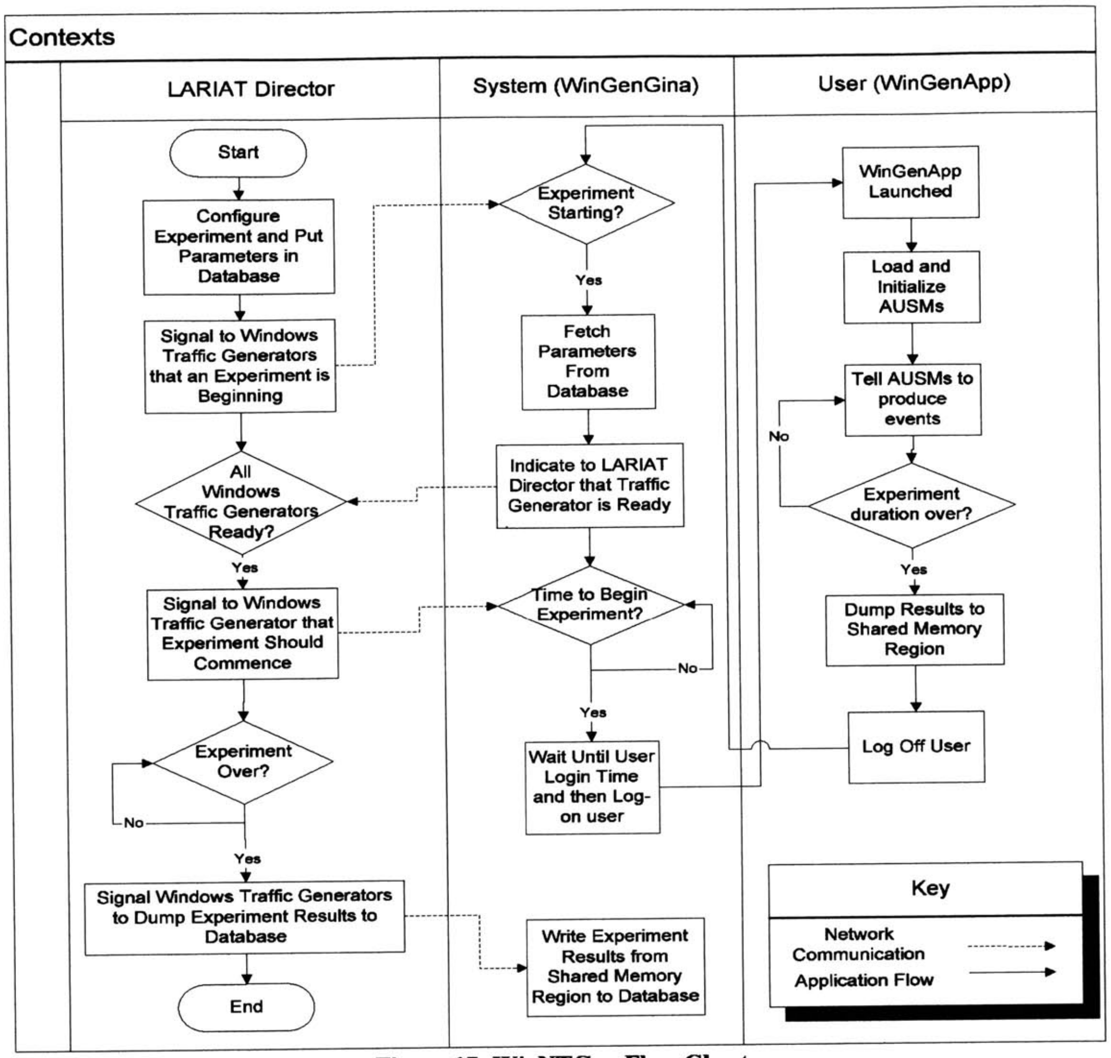

Contexts
LARIAT Director
System (WinGenGina)
User (WinGenApp)
Start
Configure Experiment and Put Parameters in Database
Signal to Windows Traffic Generators that an Experiment is Beginning
All Windows Traffic Generators Ready?
Yes
Signal to Windows Traffic Generator that Experiment Should Commence
Experiment Over?
No
Yes
Signal Windows Traffic Generators to Dump Experiment Results to Database
End
Experiment Starting?
Yes
Fetch Parameters From Database
Indicate to LARIAT Director that Traffic Generator is Ready
Time to Begin Experiment?
No
Yes
Wait Until User Login Time and then Log-on user
Write Experiment Results from Shared Memory Region to Database
WinGenApp Launched
Load and Initialize AUSMs
Tell AUSMs to produce events
No
Experiment duration over?
Yes
Dump Results to Shared Memory Region
Log Off User
Key
Network Communication
Application Flow

Figure 17: WinNTGen Flow Chart

4.3.2. System Objects

There are several important objects in the WinNTGen system that appear as a part of both WinGenGina and the WinGenApp and provide an abstraction for key, common functionalities. Each of these objects resides locally on each WinNTGen traffic generator.

The first object is the Run Log. An instance of this object is passed to all other objects in the system upon their creation so that they can make reports about the experiment and any problems encountered. The Run Log object maintains a single log, common to all objects and synchronizes access to it. At the end of the experiment, the Run Log object's contents are sent to the LARIAT database.

The next object is the Parameter Database object. A copy of this object is passed to almost all of the other objects in the system. Its role is to obtain configuration settings from the local machine as well as store retrieved parameters from the LARIAT database on the network. It maps disparate parameter sources into one namespace for convenience and abstraction. Given a context and a variable name, the Parameter Database object will return the appropriate parameter. Context refers to which category the parameter falls under. Typical categories include *Host Parameters*, *User Parameters*, etc. By having such a database locally, AUSMs will not have to be rewritten when the format of the data changes in either the LARIAT database on the network or on the local host.

As mentioned before, AUSMs are used to drive the applications. In order to manage these AUSMs as well as report statistics and information about them, a Module Manager object is used. This object allows for easy enumeration and manipulation of the loaded AUSMs.

4.3.3. WinGenGina

As depicted in Figure 18, when the Windows host is booted, the OS loads the WinGenGina replacement graphical identification and authentication DLL (GINA)[1]. Upon being loaded, WinGenGina loads the original MSGina.dll and maps all the function entry points to its locally defined function pointers. It also set up a listening TCP port in order to receive commands from the LARIAT director[2]. Until the particular Windows host is contacted by the LARIAT director, WinGenGina will behave like the original MSGina by transparently passing all calls from the WinLogon process to MSGina.

---

[1] This is done via modification of the registry value stored under HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon

[2] Currently, no authentication is being performed to verify the source of the connection. Deployable versions will need to have authentication to prevent a potential security hole.

From a GUI perspective, on the user logon screen alongside the normal logon dialog there is another WinNTGen dialog box present that provides status information as well as letting the experimenter know that WinNTGen is installed on the system.

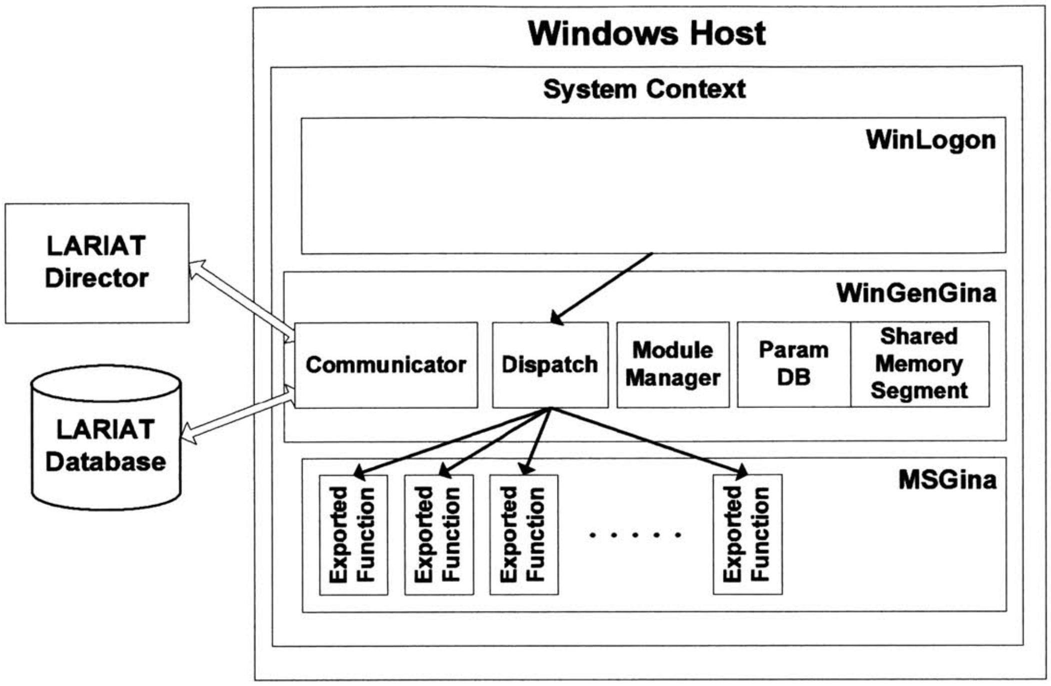


Figure 18: WinGenGina Components

When an experiment begins, the LARIAT director contacts each Windows host to communicate the IP address of the LARIAT database to use. The WinGenGina then fetches the parameters from the specified database on the network, including the user profile, host specific parameters, and specific parameters requested by the AUSMs loaded by the Module Manager. It then stores them in the Parameter Database object. After this is done, WinGenGina replies that it is ready. When all of the hosts in the testbed are ready for the experiment, the director informs them that the experiment is starting. WinGenGina then waits to the appropriate log in time of the simulated user as dictated by the user profile. When the appropriate time arrives, WinGenGina disables the interactive login and sends a custom secure attention sequence (SAS) to log in the simulated user. Before the user is actually logged in, the Parameter Database is serialized into a shared memory segment so that it can be read by the WinGenApp. Upon the simulated user being logged-in, the WinGenApp is launched[3], which then assumes control of the user simulation and traffic generation.

4.3.4. WinGenApp

The WinGenApp is responsible for driving applications according to the simulated user's profile. As with the WinGenGina, a copy of the application must be installed on every Windows NT machine taking part in the experiment. As mentioned previously, it creates network traffic by controlling applications that utilize the network. It can be extended through AUSMs that allow it to control more applications. A detail diagram of the WinGenApp appears in Figure 19.

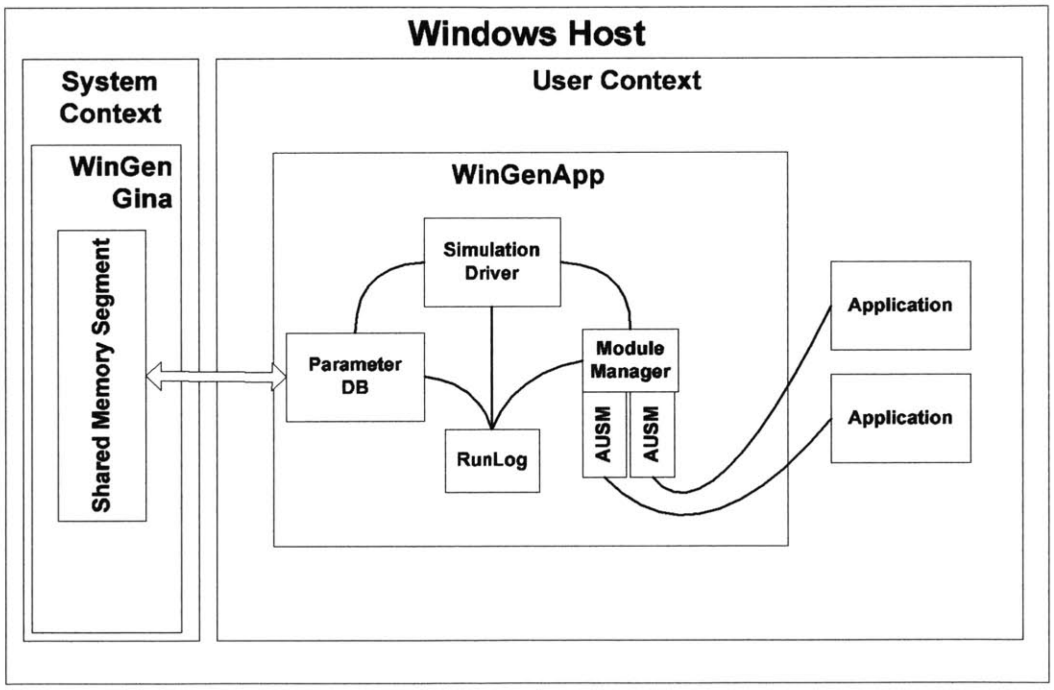


Figure 19: WinGenApp Module Diagram

[3] This is done by placing the WinGenApp in the startup folder of the simulated user.

A complete course of execution for WinGenApp during an experiment is as follows. After the simulated user is logged-in by the WinGenGina, the WinGenApp is launched. WinGenApp creates an instance of the Parameter Database by reading the from the shared memory segment created by the WinGenGina. It then creates an instance of the module manager which loads the available AUSMs on the system and prepares them for use. The complete process of AUSM selection and event generation was detailed in Section 3.3. When it is time to logoff, WinGenApp simply ends the current user session and logs off.

The stream of generated user events is recorded by WinGenApp since the AUSMs return an opaque description of each event executed. This stream may be replayed by handing back the events one at a time to the appropriate AUSM for execution. Note that during playback the individual AUSMs will not be able to make alternate transitions based on error conditions or configuration changes since the stream is predetermined. However, playback is useful when there is a phenomenon in the experiment that the experimenter would like to try to reproduce.

4.3.5. AUSM Implementation

AUSMs are implemented as dynamic link libraries (DLLs). Each DLL exports a set of functions that enables it to be used by WinGenApp as described in Table 4. Note that the return type of each function is `void` since errors are reported using structured exception handling.

Function	Description
`GetModuleInfo`	Return information about the AUSM including a description and the types of parameters it will need from the database
`Initialize`	Verify that relevant applications are present, initialize data structures, retrieve parameters from the local parameter database, and perform all general start up operations
`Generate`	Generate a series of events according to the internal model of the module
`Update`	Reread relevant parameters from the local parameter database object
`Reset`	Module should reset to the start state and prepare to be run again
`Execute`	Take the passed in event and execute it
`DescEvent`	Return a textual description of the passed in event
`DeleteEvent`	Free memory used by this event

Table 4: Exported DLL Functions

The DLLs may link into WinNTGen libraries to have access to convenient classes such as a value generator for the hybrid distribution discussed in 3.5.

Chapter 5

WinNTGen Evaluation

5.1 Overview

In order to evaluate various performance characteristics of the WinNTGen system, a network was set up as depicted in Figure 20. In order to isolate the traffic produced by WinNTGen and thereby produce a more meaningful evaluation some elements of a typical LARIAT testbed, such as the Linux network traffic generators, are not present.

The test network, testbed.edu, consists of four hosts connected by a hub. The first host, Director.testbed.edu, is a LARIAT director and database in order to provide the necessary framework for the evaluation. The second host, GenXP.testbed.edu, is running WinNTGen on Windows XP (NT 5.1) Pentium III with 640 MB of RAM. The third host, Server.testbed.edu, is the domain controller for the testbed.edu domain and is configured with a local DNS, Active Directory, Exchange Server, and several network shares. The fourth host is a sniffer and was responsible for recording all network traffic for the testbed.edu network that was used in the analyses. An external router/DNS forwarder was used so that the Internet Explorer AUSM could browse real web pages thereby permitting a better evaluation.

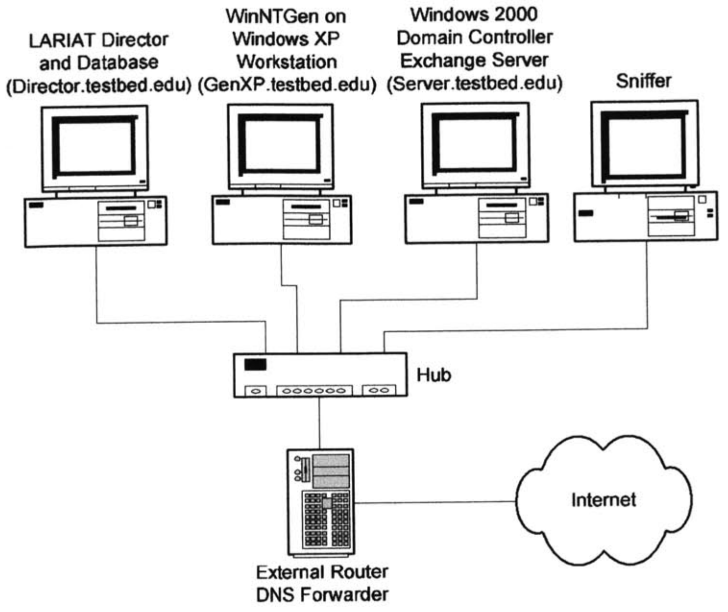

WinNTGen on Windows XP Workstation (GenXP.testbed.edu)
LARIAT Director and Database (Director.testbed.edu)
Windows 2000 Domain Controller Exchange Server (Server.testbed.edu)
Sniffer
Hub
Internet
External Router DNS Forwarder

Figure 20: Evaluation Testbed

Two different analyses were performed in order to evaluate the WinNTGen system. The first analysis verifies that the contents of the packets produced by WinNTGen are reasonably realistic. The second analysis examines system throughput.

5.2 Realism

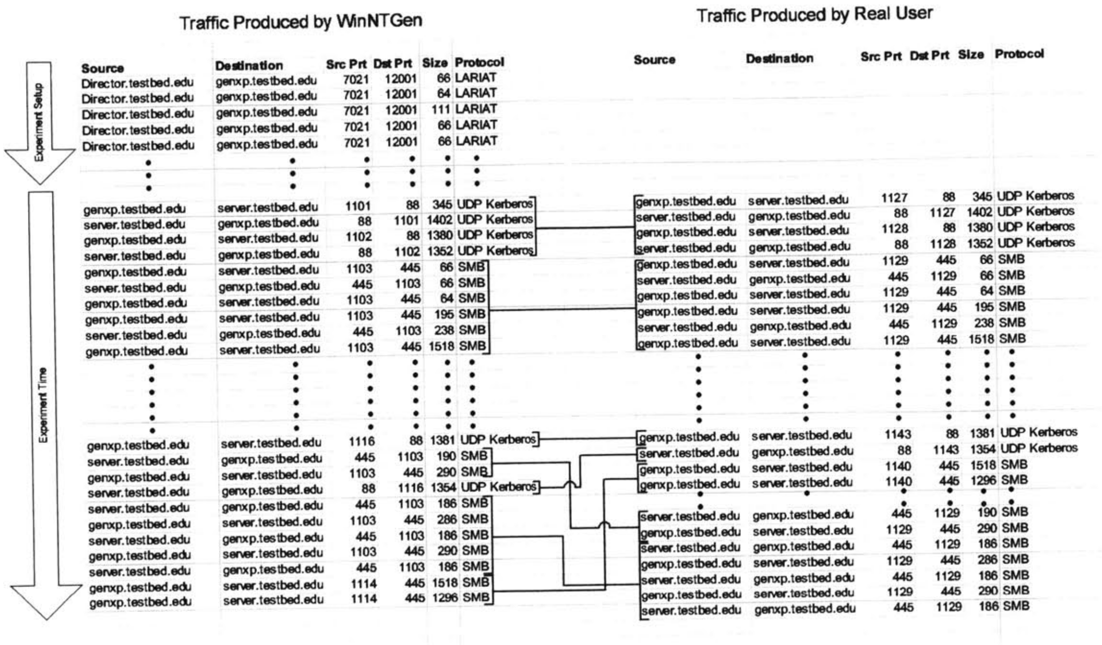
The realism of the WinNTGen system was evaluated by comparing network traffic produced by the stream of artificially generated events to the network traffic produced by a real user executing the same series of events.

Specifically, the sniffer recorded network traffic as WinNTGen executed a login sequence for a simulated user, then produced different series of user events via the different AUSMs for a four hour period, and finally logged out the simulated user. Subsections of this traffic were compared to network traffic recordings resulting from the same sequences of user events executed by a human user. The results appear in the following sections.

5.2.1. Login and Logout traffic

There should be no difference in network traffic resulting from WinNTGen simulated user logins and real user logins due to the way in which WinNTGen implements this functionality. When logging in a simulated user, WinGenGina enters the username and password of the simulated user directly into the login dialog box via the `FindWindowEx` and `SetText` functions. WinLogon is not able to differentiate between this programmatically controlled login and a real login and therefore the resulting network traffic should appear to be the same as compared to a login initiated by a real user.

A side-by-side comparison of packets recorded from the WinNTGen login and the real user login, Figure 21, reveals that the two are indeed effectively the same. In the figure there are three packet sequences. The top one is from the LARIAT Director communicating with WinGenGina. The middle one shows a case in which a WinNTGen logon produced the same traffic as a real user logon. Finally, the last sequence shows a case in which there are several overlapping conversations. The differences between the two sides in the last sequence results from the packets from different startup operations, such as downloading the user's profile, downloading the policy settings, and calls to the MS NT Directory DRS Endpoint, being interspersed differently. The WinNTGen and real user logout sequences exhibit this type of variability as well.


Experiment Setup
Experiment Time

Traffic Produced by WinNTGen

Source	Destination	Src Prt	Dst Prt	Size	Protocol
Director.testbed.edu	genxp.testbed.edu	7021	12001	66	LARIAT
Director.testbed.edu	genxp.testbed.edu	7021	12001	64	LARIAT
Director.testbed.edu	genxp.testbed.edu	7021	12001	111	LARIAT
Director.testbed.edu	genxp.testbed.edu	7021	12001	66	LARIAT
Director.testbed.edu	genxp.testbed.edu	7021	12001	66	LARIAT
•	•	•	•	•	•
genxp.testbed.edu	server.testbed.edu	1101	88	345	UDP Kerberos
server.testbed.edu	genxp.testbed.edu	88	1101	1402	UDP Kerberos
genxp.testbed.edu	server.testbed.edu	1102	88	1380	UDP Kerberos
server.testbed.edu	genxp.testbed.edu	88	1102	1352	UDP Kerberos
genxp.testbed.edu	server.testbed.edu	1103	445	66	SMB
server.testbed.edu	genxp.testbed.edu	445	1103	66	SMB
genxp.testbed.edu	server.testbed.edu	1103	445	64	SMB
genxp.testbed.edu	server.testbed.edu	1103	445	195	SMB
server.testbed.edu	genxp.testbed.edu	445	1103	238	SMB
genxp.testbed.edu	server.testbed.edu	1103	445	1518	SMB
•	•	•	•	•	•
genxp.testbed.edu	server.testbed.edu	1116	88	1381	UDP Kerberos
server.testbed.edu	genxp.testbed.edu	445	1103	190	SMB
genxp.testbed.edu	server.testbed.edu	1103	445	290	SMB
server.testbed.edu	genxp.testbed.edu	88	1116	1354	UDP Kerberos
server.testbed.edu	genxp.testbed.edu	445	1103	186	SMB
genxp.testbed.edu	server.testbed.edu	1103	445	286	SMB
server.testbed.edu	genxp.testbed.edu	445	1103	186	SMB
genxp.testbed.edu	server.testbed.edu	1103	445	290	SMB
server.testbed.edu	genxp.testbed.edu	445	1103	186	SMB
genxp.testbed.edu	server.testbed.edu	1114	445	1518	SMB
genxp.testbed.edu	server.testbed.edu	1114	445	1296	SMB

Traffic Produced by Real User

Source	Destination	Src Prt	Dst Prt	Size	Protocol
genxp.testbed.edu	server.testbed.edu	1127	88	345	UDP Kerberos
server.testbed.edu	genxp.testbed.edu	88	1127	1402	UDP Kerberos
genxp.testbed.edu	server.testbed.edu	1128	88	1380	UDP Kerberos
server.testbed.edu	genxp.testbed.edu	88	1128	1352	UDP Kerberos
genxp.testbed.edu	server.testbed.edu	1129	445	66	SMB
server.testbed.edu	genxp.testbed.edu	445	1129	66	SMB
genxp.testbed.edu	server.testbed.edu	1129	445	64	SMB
genxp.testbed.edu	server.testbed.edu	1129	445	195	SMB
server.testbed.edu	genxp.testbed.edu	445	1129	238	SMB
genxp.testbed.edu	server.testbed.edu	1129	445	1518	SMB
•	•	•	•	•	•
genxp.testbed.edu	server.testbed.edu	1143	88	1381	UDP Kerberos
server.testbed.edu	genxp.testbed.edu	88	1143	1354	UDP Kerberos
genxp.testbed.edu	server.testbed.edu	1140	445	1518	SMB
genxp.testbed.edu	server.testbed.edu	1140	445	1296	SMB
•	•	•	•	•	•
server.testbed.edu	genxp.testbed.edu	445	1129	190	SMB
genxp.testbed.edu	server.testbed.edu	1129	445	290	SMB
server.testbed.edu	genxp.testbed.edu	445	1129	186	SMB
genxp.testbed.edu	server.testbed.edu	1129	445	286	SMB
server.testbed.edu	genxp.testbed.edu	445	1129	186	SMB
genxp.testbed.edu	server.testbed.edu	1129	445	290	SMB
server.testbed.edu	genxp.testbed.edu	445	1129	186	SMB

Figure 21: Comparison of Logon Network Traffic

5.2.2. AUSM Generated Traffic

It was hypothesized that there would be slight differences between the WinNTGen and the user generated network traffic for the same sequence of user actions because of the granularity at which the existing AUSMs choose to operate. For example, when composing an email, the Outlook AUSM currently does not model the times between resolving a recipient's email address with the Exchange server, composing a message, and finally sending the email. Rather, when arriving at the *Send Email* state, it performs these actions in under a second and does not produce any traffic that results from interacting with the UI at a key stroke by key stroke basis such as auto-completion of the recipient's name as it is being typed. A real user performs the actions sequentially with delays between them.

Given that these types of differences exist and come directly as a result of model granularity, a more useful analysis is to expose any errors directly resulting from choosing to control applications through automation. Note that it is not meaningful to compare the user events generated by WinNTGen to that of a real user since the user models in use are not predictive[38], nor is it meaningful to test for self similarity[39] in the traffic [32, 40] since the timing in the AUSMs are generated by a heavy tailed process and therefore self similar by definition.

In order to generate network traffic for analysis, WinNTGen was run for a four hour period with the Internet Explorer, Windows Explorer, and Outlook AUSMs loaded. The resulting network traffic was recorded and then filtered so that only traffic produced directly as a result of WinNTGen was retained. From this recorded traffic, three random segments were chosen that each contained traffic produced mostly by one AUSM. Any traffic not produced by the AUSM in interest was filtered out. The chosen segments appear in Figure 22, Figure 23, and Figure 24.

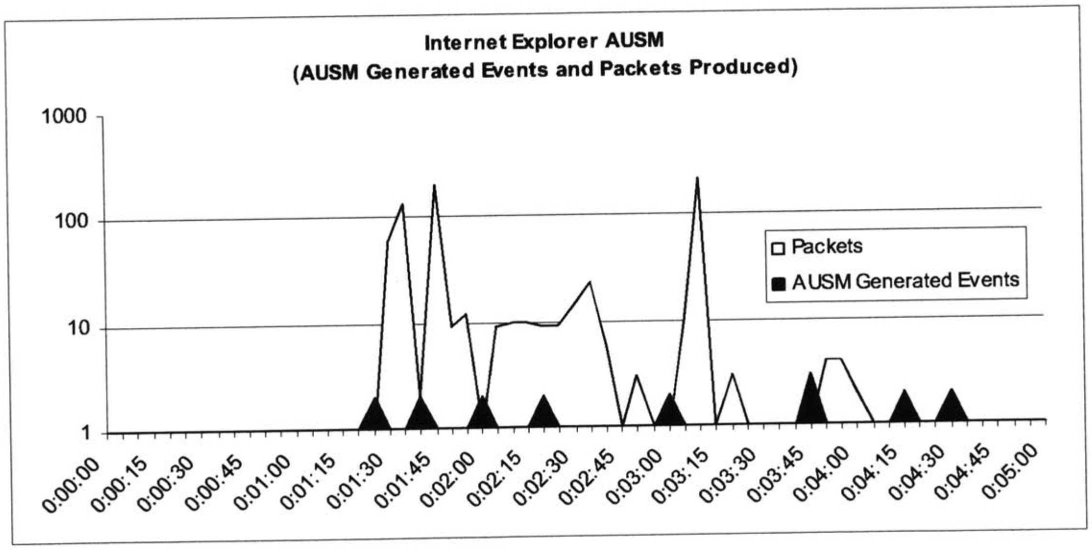


Figure 22: Internet Explorer AUSM (AUSM Generated Events and Packets Produced)

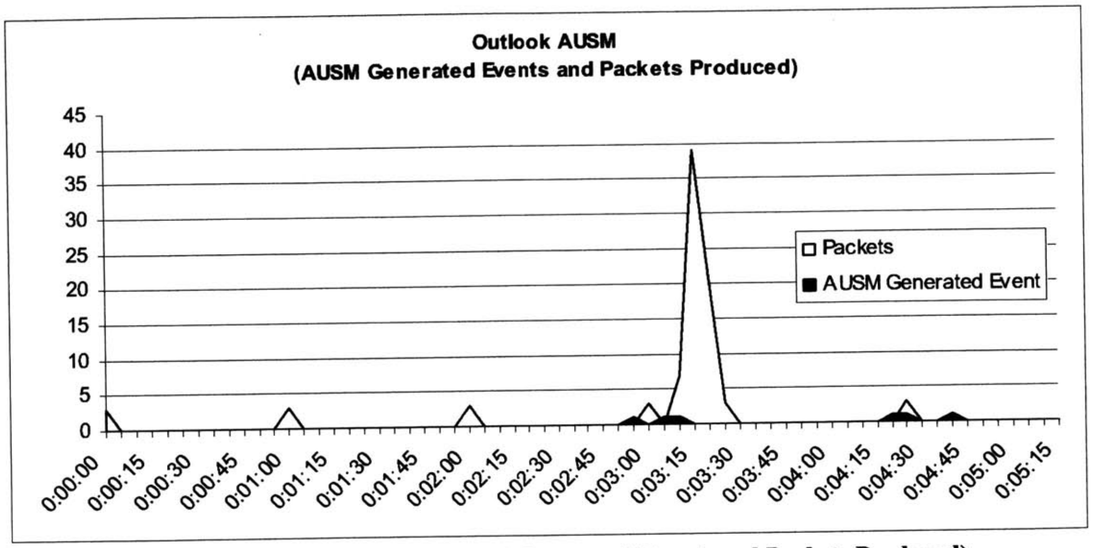


Figure 23: Outlook AUSM (AUSM Generated Events and Packets Produced)

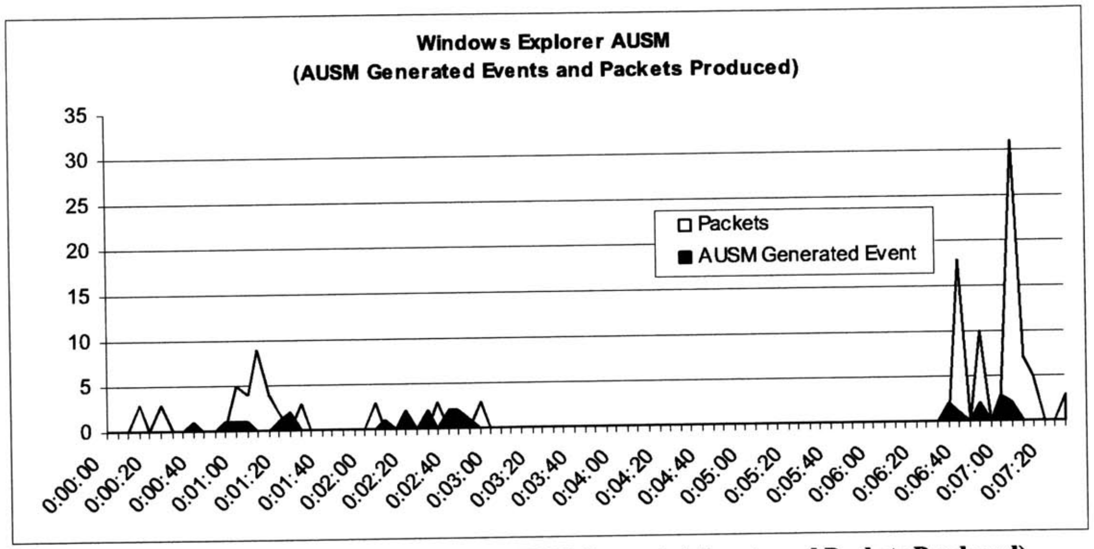


Figure 24: Windows Explorer AUSM (AUSM Generated Events and Packets Produced)

There are several aspects of the traffic that are worth noting. In Figure 22, as expected, whenever a user event occurs, such as navigating to a hyperlink in a document, the browser produces large bursts of packets as the new page and all its embedded references (e.g. images) are retrieved. In Figure 23, the packets that are being produced without user action are from Outlook periodically checking the simulated user's inbox. Also in this figure is a large spike of packets around the 3:15 mark that result from the simulation user deciding to download the actual messages. Finally, in Figure 24, the correlation between user browsing events and packets produced can be seen.

It is apparent that some user actions produce no traffic. For example, in the case of Internet Explorer, cached web pages are usually not downloaded again when the user browses back to them. Also, the action of closing Windows Explorer produces no traffic. These events should still be included, however, since they affect the OS state, and it is possible that in the future these events will produce packets on the network.

5.3 Throughput

Even though the goal of the WinNTGen system is to produce realistic network traffic, it is also necessary in some experiments to be able to produce large amounts of traffic of a certain type in order to stress test network services or, more generally, in cases where realism is not the primary goal.

To test the throughput of WinNTGen on a single machine, the delays between simulated user events were removed and the AUSMs were tested independently for increasing numbers of running instances of WinNTGen. In these tests, the actual data rate was not considered since it is based completely on content such as the size of the emails, and does not meaningfully measure the performance of WinNTGen. Rather, the tests measured the packet producing ability of each AUSM. The results of this test appear in Figure 25.

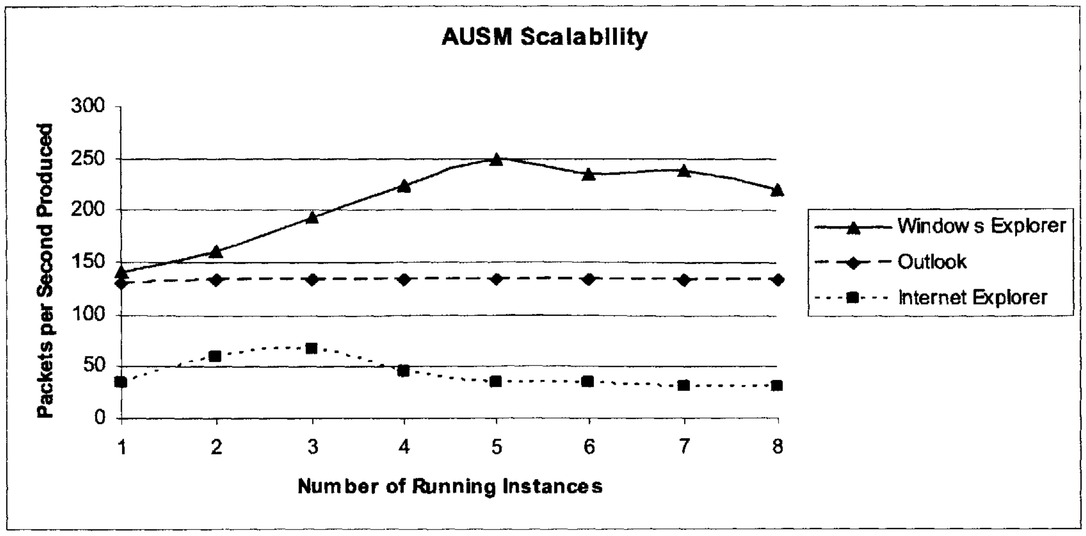


Figure 25: AUSM Scalability

For the Outlook AUSM there is a constant traffic rate regardless of the number of instances of WinNTGen. This can be explained by the fact that each Outlook AUSM instance is controlling the same instance of Outlook. Since a single AUSM instance is able to completely utilize Outlook, adding additional instances does not increase the rate at which Outlook can perform tasks.

For the Internet Explorer AUSM there are initial increasing returns in packet rates to the number of instances of the AUSM since each AUSM controls a different instance of Internet Explorer. However, as more instances are added they begin to compete with one another for local resources and the local machine begins to thrash.

Finally, for the Windows Explorer AUSM, an increase in packet rate was observed as the number of instances of the AUSM was increased. However, eventually there is are decreasing returns to adding more instances of WinNTGen as the separate instances begin to compete for local system resources as the Internet Explorer AUSMs did.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

It is clear that network based IA systems need to be thoroughly evaluated before they are deployed so that their true performance may be ascertained. To do this, they must be examined in controlled, evaluation network environments that are representative of actual production network environments. Due to the ubiquity of hosts running Windows OSs, it is necessary to include network traffic generated by Windows hosts in these evaluations. WinNTGen was shown to be a tool capable of producing such realistic network traffic, as well as being able to integrate into the existing LARIAT framework.

Through modeling user behavior at an application and session layer, WinNTGen is capable of producing complex, realistic network traffic though simple models of application use. By controlling applications via AUSMs, WinNTGen is modular and easily extensible.

In implementing WinNTGen, it became evident that modifying and hooking into the Windows NT OS at the kernel level in order to create multiple virtual hosts on one physical host is time consuming and difficult and still does not result in the production of realistic network traffic. Clearly, a better approach was to assume one user per physical host, to perform user modeling instead of application and protocol modeling, and to allow the Windows NT OS to perform much of the network traffic generation through its normal activity.

6.2 Future Work

There are several ways in which WinNTGen can be further developed. In order to add variety to the generated traffic, more AUSMs can be added to the system in order to increase the number of applications that could be controlled.

Also, the AUSMs can be made more complex in order to produce chains of events that more closely resemble those produced by a human. There are many improvements that can be made in this area, but there are a couple of key ones.

First, by adding daily goals for the user to achieve, sequences of user events would appear to have a purpose. That way, when the simulated user went web surfing, he would tend to follow links more related to what he needs and would send emails that are more relevant to the daily goal. For a network based IA system looking for suspicious browsing behavior, this might be important. These systems may flag suspicious activities such as a user performing web searches for the word "sploits" and browsing to sites known to host malicious code.

Second, by adding requires and provides concepts to the modules, a control interface could be used for classes of applications. For example, web browsers and email clients could be controlled without dependence on the actual client. This way, for example, a simulated user could be browsing and choose to forward a URL to a friend with the user's favorite email client.

WinNTGen already provides limited modeling of user interaction with a Windows user interface and the network traffic it produces is realistic. Its architecture and implementation will support these and other additions.

References

[1] "IEEE Task Force on Information Assurance," http://www.ieee-tfia.org. 2002.

[2] "2002 CSI/FBI Computer Crime and Security Survey," in *Computer Security Issues and Trends*, vol. 3: Computer Security Institute, 2002, pp. 24.

[3] S. Northcutt, *Network Intrusion Detection: An Analyst's Handbook*, 2nd ed: New Rider's Publishing, 1999.

[4] R. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, and M. A. Zissman, "Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation. Proceedings DARPA Information Survivability Conference and Exposition," Hilton Head, SC, USA, 1999.

[5] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "Analysis and results of the 1999 DARPA off-line intrusion detection evaluation. Recent Advances in Intrusion Detection," Toulouse, France, 2000.

[6] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 DARPA off-line intrusion detection evaluation," *Computer Networks*, vol. 34, pp. 579-95, 2000.

[7] J. W. Haines, L. M. Rossey, R. P. Lippmann, and R. K. Cunningham, "Extending the DARPA off-line intrusion detection evaluations. Proceedings DARPA Information Survivability Conference and Exposition II," Anaheim, CA, USA, 2001.

[8] L. M. Rossey and R. K. Cunningham, "LARIAT: Lincoln Adaptable Real-time Information. Assurance Testbed," presented at IEEE Aerospace, Big Sky, Montanna, 2002.

[9] R. Marty, "Thor: A Tool to Test Intrusion Detection Systems by Variation of Attacks," Diploma Thesis. IBM Zurich Research Laboratory, 2002.

[10] D. Newman, J. Snyder, and R. Thayer, "Crying wolf: False alarms hide attacks," Network World Fusion. http://www.nwfusion.com/techinsider/2002/0624security1.html. 2002.

[11] N. J. Puketza, K. Zhang, M. Chung, B. Mukherjee, and R. A. Olsson, "A methodology for testing intrusion detection systems," *IEEE Transactions on Software Engineering*, vol. 22, pp. 719-29, 1996.

[12] A. Seleznyov, O. Mazhelis, and S. Puuronen, "Learning temporal regularities of user behavior for anomaly detection. Information Assurance in Computer Networks," presented at Information Assurance in Computer Networks: Methods, Models, and Architectures for Network Security, International Workshop, St. Petersburg, Russia, 2001.

[13] G. Shipley, "ISS RealSecure Pushes Past Newer IDS Players," in *Network Computing*, 1999.

[14] W3Schools.com, "Browser Statistics," http://www.w3schools.com/browsers/browsers_stats.asp. 2002.

[15] TheCounter.Com, "OS Stats," http://www.thecounter.com/stats/2002/April/os.php. 2002.

[16] "Google Zeitgeist - Search patterns, trends, and surprises according to Google," http://www.google.com/press/zeitgeist.html. 2002.

[17] BizNix, "The Business Web Server Survey," http://www.biznix.org/surveys/. 2000.

[18] "eTest suite: Web Test + Monitoring," http://www.empirix.com/Empirix/web+test+monitoring/products/e-test+suite.html. 2002.

[19] "Smbtorture, the Samba suite stress testing tool," http://www.samba.org. 2002.

[20] "InterWorking Labs, Inc: SNMP Agent," http://www.iwl.com/Resources/Papers/winsnmp.html. 2002.

[21] "Sunrise Telecom: LanExplorer," http://www.sunrisetelecom.com/lansoftware/lanexplorer.shtml. 2002.

[22] "LanTraffic V2: IP Traffic Test & Measure," http://www.zti-telecom.com/pages/iptraffic-test-measure.htm. 2002.

[23] H. Hlavacs and G. Kotsis, "Modeling user behavior: a layered approach," presented at 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, College Park, MD, USA, 1999.

[24] T. Lau and E. Horvitz, "Patterns of search: analyzing and modeling Web query refinement," presented at Seventh International Conference on User Modeling, Springer Wien, Austria, 1999.

[25] "National Institute of Standards and Technology (NIST) Expect," http://expect.nest.gov. 2001.

[26] S. K. Card, T. P. Moran, and A. Newell, *The psychology of human-computer interaction.* Hillsdale, NJ: Lawence Erlbaum Associates, 1983.

[27] B. E. John, "The GOMS Family of User Interface Analysis Techniques: Comparison and Contrast," *ACM Transactions on Computer-Human Interaction*, vol. 3, pp. 320-351, 1996.

[28] D. Kupper and A. Kobsa, "User-tailored plan generation," presented at Seventh International Conference on User Modeling, Springer Wien, Austria, 1999.

[29] D. Rogerson, *Inside COM.* Seattle, Washington: Microsoft Press, 1997.

[30] Microsoft, "Handling HTML Element Events," http://msdn.microsoft.com/library/default.asp?url=/workshop/browser/mshtml/tutorials/sink.asp. 2002.

[31] L. Jung Jin and R. McCartney, "Predicting user actions using interface agents with individual user models," presented at Second Pacific Rim International Workshop on Multi-Agents, Kyoto, Japan, 1999.

[32] V. Paxson and S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modeling," *IEEE\ACM Transactions on Networking*, vol. 3, pp. 226-244, 1995.

[33] Wie Hua, Jim Ohlund, and B. Butterklee, "Unraveling the Mysteries of Writing a Winsock 2 Layered Service Provider," http://www.microsoft.com/msj/defaultframe.asp?page=/msj/0599/layeredservice/layeredservice.htm&nav=/msj/0599/newnav.htm. 1999.

[34] MSDN, "Windows Sockets 2," http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winsock/apistart_9g1e.asp?frame=true.

[35] "Multihomed Issues with Windows NT (Q181774)," Microsoft Product Support. http://support.microsoft.com/default.aspx?scid=kb;EN-US;q181774.

[36] J. Kohl and C. Neuman, "The Kerberos Network Authentication Service (V5)," http://www.faqs.org/rfcs/rfc1510.html. 1993.

[37] "VMWare -- Virtual Computing Throughout The Enterprise," http://www.vmware.com. 2002.

[38] N. Lesh, C. Rich, and C. L. Sidner, "Using plan recognition in human-computer collaboration," presented at Seventh International Conference on User Modeling, Springer Wien, Austria, 1999.

[39] J. Beran, "Statistics on Long-Memory Processes," in *Monographs on Statistics and Applied Probability*. New York, NY: Chapman and Hall, 1994.

[40] Mark Crovella and A. Bestavros, "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes," presented at ACM SIGMETRICS Conference, Boston, MA, 1996.