ABSTRACT

SOFT FAULT DETECTION USING MIBs IN COMPUTER NETWORKS

by
Sachin Arora

To improve network reliability and management in today's high-speed communication system, a statistical anomaly network intrusion detection system (NIDS) has been investigated, for network soft faults using the Management Information Base (MIB) traffic parameters provided by Simple Network Management Protocol (SNMP), for both wired and wireless networks. The work done would be a contribution to a system to be designed MIB Anomaly Intrusion Detection, a hierarchical multi-tier and multi-observation-window Anomaly Intrusion Detection system. The data was derived from many experiments that had been carried out in the test bed that monitored 27 MIB traffic parameters simultaneously, focusing on the soft network faults. The work here has been focused on early detection, i.e., detection at low values of the ratio of fault to background traffic. The performance of this system would be measured using traffic intensity scenarios, as the fault traffic decreased from 10% to 0.5% of the background.

SOFT FAULT DETECTION USING MIBs IN COMPUTER NETWORKS

by
Sachin Arora

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
In Partial Fulfillment of the Requirements for the Degree in
Master of Science in Computer Engineering

Department of Computer Engineering

May 2003

APPROVAL PAGE

SOFT FAULT DETECTION USING MIBs IN COMPUTER NETWORKS

Sachin Arora

Dr. Constantine Manikopoulos, Thesis Advisor — Date
Associate Professor, Department Electrical and Computer Engineering, NJIT

Dr. George Antoniou, Committee Member — Date
Professor, Department of Computer Science, Montclair State University

Dr. Bin He, Committee Member — Date
Senior scientist, XPRT Solutions Inc.

BIOGRAPHICAL SKETCH

Author: Sachin Arora

Degree: Master of Science

Date: May 2003

Undergraduate and Graduate Education:

- Master of Science in Computer Engineering
  New Jersey Institute of technology, Newark, NJ, 2003

- Bachelor of Engineering in Electronics
  University of Nagpur, India, 2000

Major: Computer Engineering

"To my Parents who are a constant source of support and encouragement."

ACKNOWLEDGEMENT

I would like to express my sincere appreciation to Dr. Constantine N. Manikopoulos for serving as a Thesis Advisor and providing the much needed directions with this research.

I would like to express my gratitude towards my family and Suvarcha Malhotra for their insurmountable support and patience.

I would like to thank my family and Suvarcha Malhotra for their insurmountable support and patience.

I used CONEX Laboratory, Room 410B Faculty Building at NJIT to do this research and would like to thank my colleagues, Ranjit Salunkhe, Olufemi Tairu, Pablito Lake and Jun Li for their cooperation and assistance throughout the entire course of this research.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

LIST OF FIGURES

Chapter **Page**

CHAPTER 1

INTRODUCTION

Today's communication network is still not very stable and reliable. Failures, hard or soft, happen every once in a while. In order to have a bright future for the network, the Internet, failures have to be dealt, soft or hard. Network intrusion detection aims to protect networks and computers from malicious network-based attacks. Intrusion detection techniques can be partitioned into two complementary trends: misuse detection, and anomaly detection. Misuse detection systems model the known attacks and scan the system for the occurrences of these patterns. Anomaly detection systems, flag intrusions by observing significant deviations from typical or expected behavior of the systems or users. The basic assumption of anomaly intrusion detection is that an intruder's behavior will be noticeably different from that of legitimate users.

This research aims at designing a detector, the MIB Anomaly Intrusion Detection (MAID) system. In defending against network faults/DOS attacks, achieving early detection and warning is of paramount significance. The corresponding experimental results demonstrate that MAID can reliably detect soft faults with traffic anomaly intensity as low as 1% of the typical background traffic intensity, far below the intensity level that causes harm, thus generating an effective early warning.

1.1 Network Management

Network management is a broad term, as defined by ISO/OSI Specific Management Functional Areas (SMFA), includes the following five functional areas: fault management, accounting management, configuration management, performance management and security management. Fault management is one of the most important aspects of network management. Fault management is the detection of a problem, fault isolation and correction to normal operation. The word "detection" is divided into three parts: (a) whether there is a fault occurring, (b) if so, what kind of fault occurs, (c) where the location of the fault is.

Traditional fault management emphasizes detection and processing hard network faults and alarms. This method is necessary but when network alarms are captured, filtered and analyzed, service and network failures are already present. Therefore, traditional fault management is more reactive in nature. Since network performance degradations are signatures of network faults and are preludes to service failures, being able to detect them early and automatically enables timely and rapid fault containment and correction, through which serious network and service failures can be avoided. This approach complements the traditional fault management.

1.2 Classification of Faults and Fault Management

1.2.1 Types of Faults

Network faults can be classified into two categories:

Hard failures, in which the networks, or some of its elements, are not able to deliver any traffic at all. It may consist of links or nodes failures. *Soft failures*, network/service anomaly or performance degradation in various performance parameters, i.e., decrease in bandwidth, increase in delay, etc.

All, the system administrators as well as the users, can easily notice a hard fault. However, defining and otherwise characterizing and detecting soft faults are difficult. Wireless networks are particularly vulnerable to soft faults in that they have much lower bandwidth than their wired counterparts, while they are more prone to overloads, noise, congestion, etc.

Once detected, hard faults are corrected. Soft faults, however, are generally tolerated in the short term and may be monitored for longer-term trends. The goal of fault management is to address both hard and soft faults to maintain consistent network performance.

1.2.2 Types of Fault Management

The ways to manage the fault can be categorized as either reactive or proactive. The reactive fault management system waits for a problem and then troubleshoots it, i.e. you react to the problem after it has occurred. The proactive management system preset critical operational thresholds, such as network utilization. This is where you are monitoring components and the system provides a root cause alarm for the problem at hand and automatic restore processes are in place where it is possible to minimize downtime. The work done here contributes towards designing a Proactive Fault Detection [12,13,14,15] system.

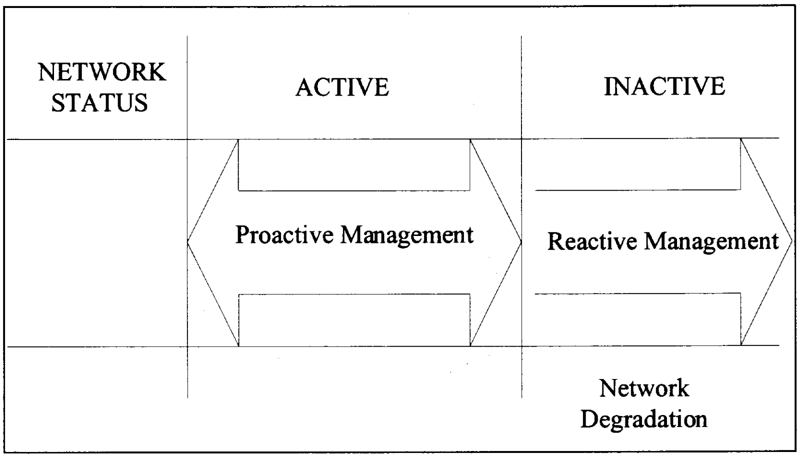


Figure 1.1 Proactive and Reactive Fault Management.

Proactive detection of network failures and performance degradations is a key to rapid fault recovery and thus robust networking, and has been receiving increasing attention lately. Proactive network performance and fault management is concerned with performance monitoring/analysis and fault detection capable of detection soft network and service faults automatically and adaptively in the midst of networks performance fluctuation and evolutions.

CHAPTER 2

SIMPLE NETWORK MANAGEMENT PROTOCOL AND MANAGEMENT INFORMATION BASE

2.1 Simple Network Management Protocol

Network management system contains two primary elements: a manager and agents. The Manager is the console through which the network administrator performs network management functions. Agents are the entities that interface to the actual device being managed. Bridges, Hubs, Routers or network servers are examples of managed devices that contain managed objects. These managed objects might be hardware, configuration parameters, performance statistics, and so on, that directly relate to the current operation of the device in question. These objects are arranged in what is known as a virtual information database, called a management information base (MIB). Simple Network Management Protocol (SNMP) [18,19] allows managers and agents to communicate for the purpose of accessing these objects. The relationship between the management station, the agent, and the MIB is shown in the following figure.

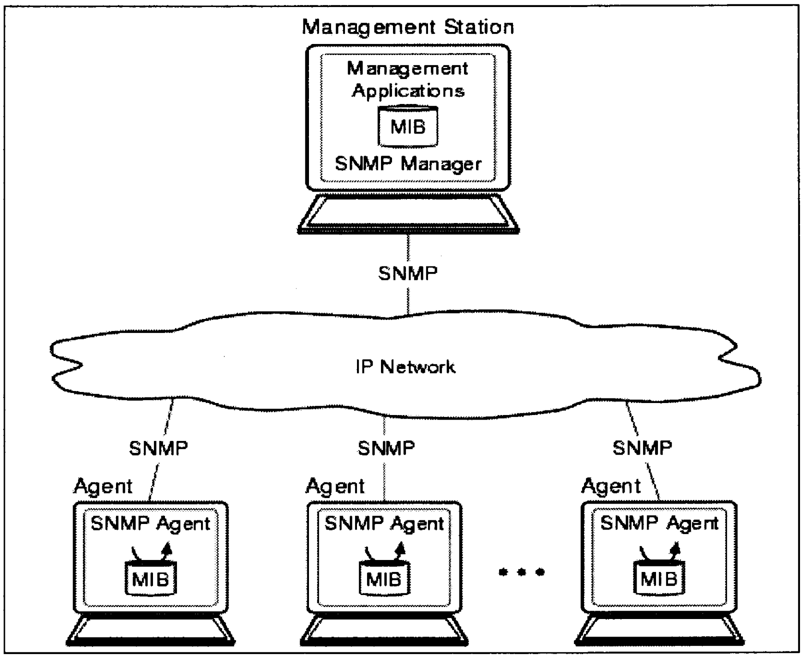


Figure 2.1 SNMP Managed Configuration.

An SNMP-compliant MIB contains definitions and information about the properties of managed resources and the services that the agents support. The manageable features of resources, as defined in an SNMP-compliant MIB, are called managed objects *or* management variables (or just objects or variables).

A management station gets and sets objects in the MIB, and an agent notifies the management station of significant but unsolicited events called traps. All message exchanges between the management station and its agents take place using the Simple Network Management Protocol (SNMP), which uses the UDP layer in the network layer stack.

Managers invoke an SNMP client on their local computer, and use the client to contact one or more SNMP servers, that run on remote machines. SNMP uses a fetch-store paradigm. A separate standard for a Management Information Base defines the set of variables that SNMP servers maintain as well as the semantics of each variable.

2.2 MIB Information Structure

The structure of management information (SMI), an SNMP standard described in the RFC 1155, defines the structure of the MIB information and the allowable data types. The SMI identifies how resources within the MIB are represented and named. The philosophy behind SMI is to encourage simplicity and extensibility within the MIB.

The SNMP specification includes a template, known as an Abstract Syntax Notation One (ASN.1) OBJECT TYPE macro, which provides the formal model for defining objects and tables of objects in the MIB.

2.2.1 MIB Object Identifiers

Each object in the MIB has an object identifier (OID), which the management station uses to request the object's value from the agent. An OID is a sequence of integers that uniquely identifies a managed object by defining a path to that object through a tree-like structure called the OID tree or registration tree. When an SNMP agent needs to access a specific managed object, it traverses the OID tree to find the object. Each one of the nodes on the tree is assigned a label consisting of an integer and a quick description.

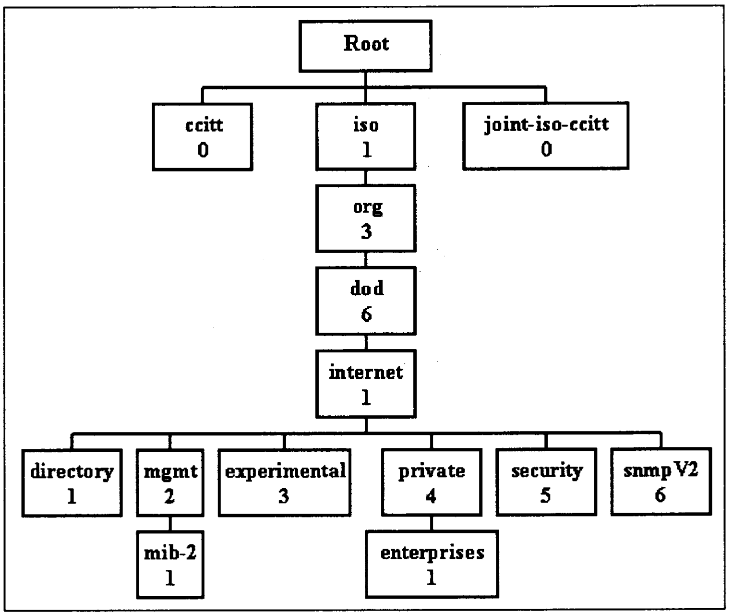


Figure 2.2 The MIB OID Tree.

The Identifiers (names) for the MIB objects are issued from this tree. The identifier of an object (the OID, Object ID) is a series of integers guiding through the tree from the root to the leaf where the object resides (for example all objects in mib-2 would start with 1.3.6.1.2.1...). The text part of each node helps to recognize what the string of number stands for.

Thus object name: iso_org_dod_internet_mgmt_mib2_interfaces_ifTable_ifEntry_ifOperStatus is easier to understand than 1.3.6.1.2.1.2.2.1.8, the same in the number format.

2.2.2 MIB-II (RFC 1213)

MIB has been developed further into MIB-2 to support more standard objects and groups, and it has it's own node on the tree. Eleven groups are referenced in the original MIB-II document (RFC 1213) [17].

One of these, CMOT (ISO Common Management Information on top of TCP/IP), is no more used because this project was abandoned. The 10 remaining groups describe the most basic information needed to manage a TCP/IP Internet.

Below is the development of the MIB Node:

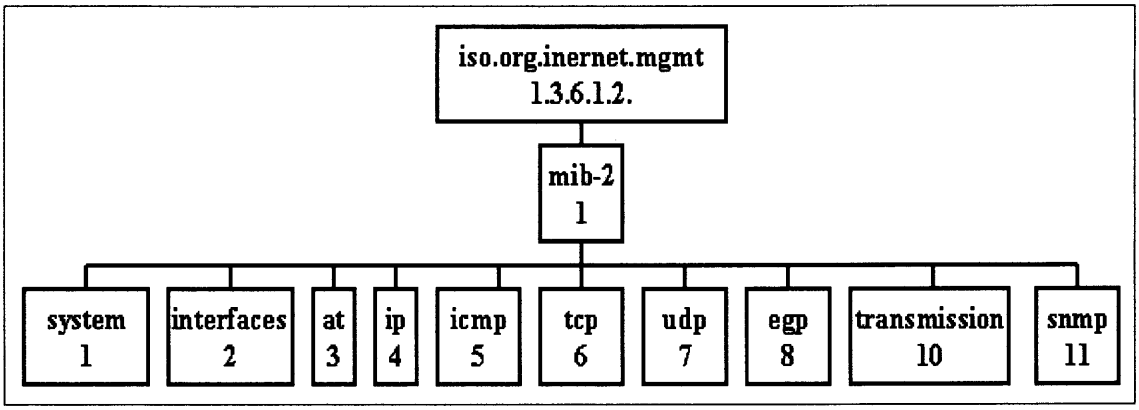


Figure 2.3 MIB-II Sub Tree.

2.3 Promise of MIB based Anomaly NID

In network intrusion detection (NID) systems, gathering the network traffic data needed to evaluate the security status of the network is a significant portion of the overall processing burden. However, many networks already deploy SNMP network management, thus MIB objects are available to collect data, in many network elements. By enlisting the MIB objects, MAID promises a lower overhead approach for analysis by the anomaly detection engine. The network manager queries the agents and retrieves the value of MIB objects to perform monitoring functions.

The advantages of this approach are:

- If an SNMP agent already is operating at the node, as is likely, the collection of local information needs little in additional resources.
- A large number of traffic related performance parameters are readily available, as needed for network intrusion detection.
- The standardized representation of the data collected in each node facilitates data exchange between nodes.
- It can be extended to collect additional data relative to network activities.
- It does not depend on the operating system.
- There are several object and parameter groups in standard MIB II that collect information on different layers and protocols, e.g., IP, ICMP, UDP, TCP, etc.

The limitation of this approach is that it is geared to handle traffic related parameters, it is basically a NID system, so it is not sufficient by itself to detect escalation of privilege, buffer overflow, etc., the type of attacks that a host intrusion detection (HID) system would detect.

CHAPTER 3

IMPLEMENTATION

3.1 Simulation of Internet Traffic in Wired Network

A stand-alone network was constructed for the purpose of carrying out actual attack experiments in a controlled environment. The network topology is shown in Figure 3.1. It comprised of four network subnets, connected by a layer 3 switch. One subnet constructed and delivered the background traffic, fashioned to follow some desired traffic, according to a chosen distribution. In these experiments, traffic with self-similar characteristics that emulated Internet type traffic was constructed, as the background traffic of a given intensity. Another subnet consisted of attack stations of various operating systems, mainly Windows and Linux. The attack and background network traffics were directed toward the victim stations that comprised another subnet. There was also a wireless victim subnet that was not utilized here. The Layer 3 switch provided the routing functionality needed for communication from one network segment to another

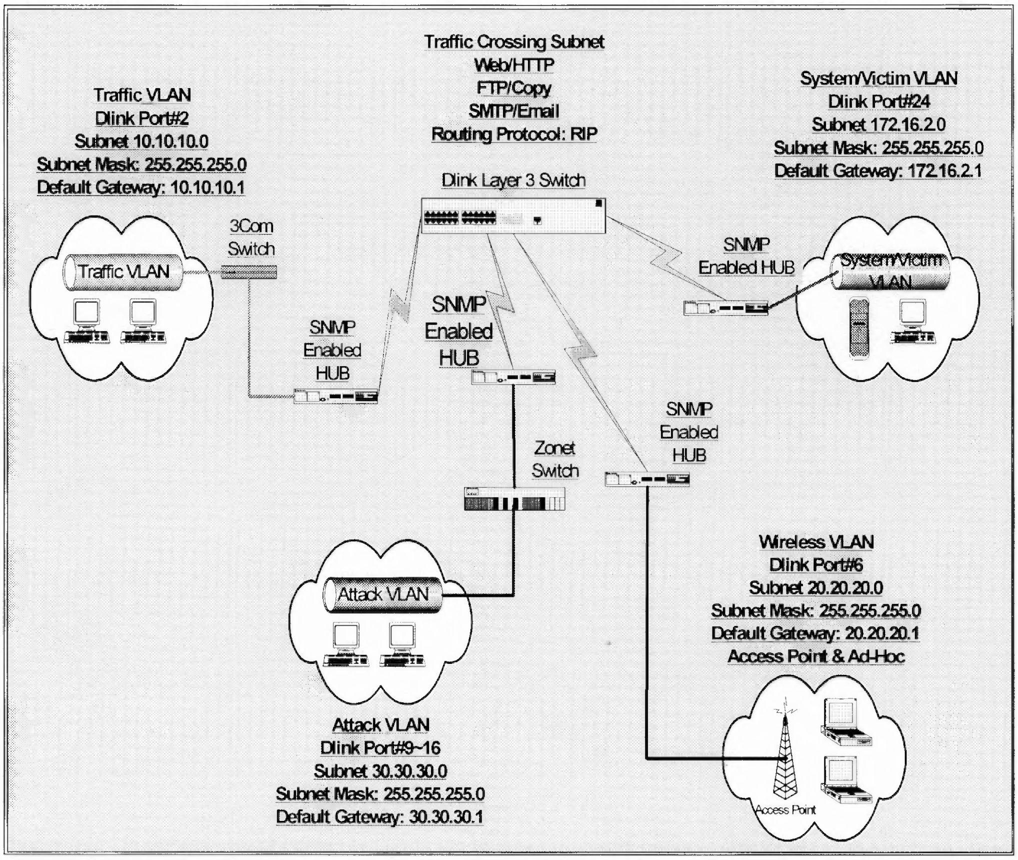


Figure 3.1 Test Wired Network.

The names of the subnets (Virtual LANs) describe by themselves what their function was in the test bed.

The Traffic segment had been set up to generate background traffic. All clients sitting in this segment generate traffic, which hit the victim segment.

The System/Victim segment had been setup to receive random patterns of predetermined traffic from both the traffic and devices residing directly within the System/Victim network.

The Wireless segment had been setup to compliment the existing setup, which had been configured. Since wireless technology is utilized in various environments, an opportunity was introduced to add the technology into the multiple networks and study various findings. In order to initiate the solution, an access point had been selected to connect into the environment.

The Attack segment had been setup to send different types of attacks including random patterns of the fault traffic.

Many actual network experiments had been carried out focusing on the Denial of Service (DOS) class of attacks, including UDP, ICMP and TCP flooding attacks. These scenarios ranged from modest (10%) to small (0.5%) values of the ratio of attack to background, R=(A/B), traffic intensity. The purpose was to evaluate the performance of MAID, as the ratio R became small, to almost insignificant levels.

3.2 Simulation of Internet Traffic in a Wireless Network, in a Wired Environment

Wireless mobile ad hoc networks differ from wired networks in that their topologies are highly dynamic and their links can have a relatively high bit error rate. These properties make it difficult to conduct controlled, repeatable experiments with routing and other protocols in a wireless ad hoc network environment. The dynamic topology switch emulates a wireless mobile ad hoc network using standard Ethernet physical connections.

Traditional switches for wired networks, such as Ethernet switches, ATM switches, and IP routers, rely on multiple access control (MAC) or Internet protocol (IP) address information to determine forwarding and the emulated connectivity cannot be altered without altering MAC or IP level addressing. Further, conventional commercial switches cannot emulate the effects of packet loss or data rate limitations. Thus, a wired network and a traditional switch cannot be used directly to emulate a mobile ad hoc network. The above problem calls for a solution where a standard wired network uses a MANET emulator [8] to create equivalent conditions that mimics a wireless ad hoc network.

3.2.1 Installation of MANET EMULATOR (Dynamic Topology Switch)

- The operating system used to build this software switch was Linux Redhat 7.0, kernel version 2.2.16.
- It had multiple network interfaces, either by using multiple interface cards and/or multiple-port interface cards/PCI Extender Card. An example block diagram of the network that may be built using this software switch is shown below.

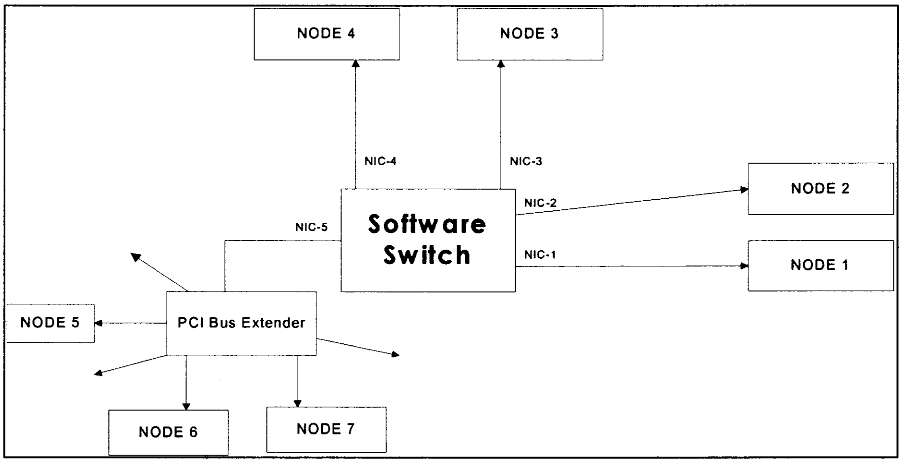


Figure 3.2 Sample network implementing the Dynamic Switch with 5 NICS

- The installation required a kernel recompilation [3].

3.2.2 Features of Dynamic Topology Switch

The dynamic topology switch emulates a MANET using standard Ethernet or other wired physical connections and requires no changes to the network's hosts.

The basic concept of operation for the dynamic topology switch is to control the connectivity of "mobile" nodes using the central hub in a star network. The dynamic topology switch can switch traffic between any set of connected hosts, based on a local switch connectivity table that can change dynamically. The switch is transparent to all the other nodes at and above the MAC layer. All incoming frames are switched based solely on the input interface and the switch connectivity table information. The switch does not alter the MAC frame or IP datagram information in anyway and, in particular, it does not add any address information of its own to the MAC frame or IP datagram. Hosts receive packets from all current neighbors, including packets not addressed to the host.

3.2.3 Dynamic Topology Switch Software

The software can be divided into three parts:

(i) User space program

(ii) Broker program

(iii) Kernel space program

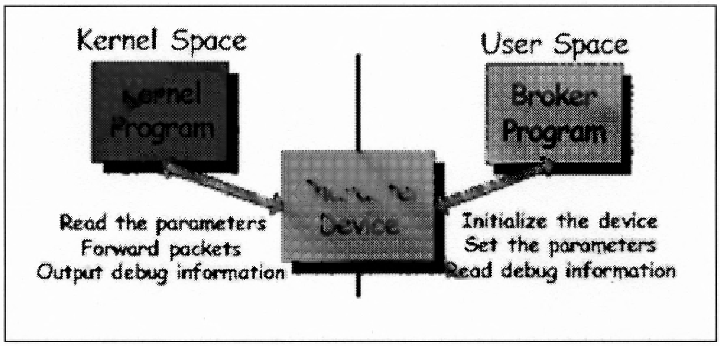


Figure 3.3 Components of the software of Dynamic Topology Switch.

The user space program is responsible for interactions with users. The kernel space program handles kernel interruptions and received packets. The broker program contains a character device driver, which is used to exchange information between user space and kernel space. The user space program first translates user inputs or command files into the proper command format. The translated commands are written into the character device in the broker program. The broker is a module that can be loaded in superuser mode. The broker creates a character device and sets all network interfaces to promiscuous mode during initialization. The broker program also maintains the switch connectivity table and token buffer queues. It continues to listen for input/output interrupts from the character device and calls the proper procedures to handle requests from the user space program. This allows users to use commands or input files to control the dynamic topology switch as a function of time. The broker is also responsible for moving outgoing packets into the proper buffers of the network devices.

The kernel space program deals with packet capture and dynamic forwarding. Once a packet is captured, the kernel procedure enters the dynamic switch block if the character device driver is loaded. The kernel space program looks up the outgoing port(s) for each incoming packet in the switch connectivity table via the broker program. The switch does not examine packets, but they are duplicated if necessary so that one incoming packet can be delivered to multiple output ports. The kernel space program forwards packets to the proper devices using the sending procedure in the broker program.

3.2.4 Mobile Network Parameters that can be Emulated Using the Dynamic Topology Switch

(I) Packet drop Rate

Mobile ad hoc networks are implemented using wireless communications where packet drops due to bit errors may be likely. In the dynamic topology switch, the packet drop rate for each connected channel can be controlled. It uses a two-state discrete-time Markov model, for packet drops, as shown below.

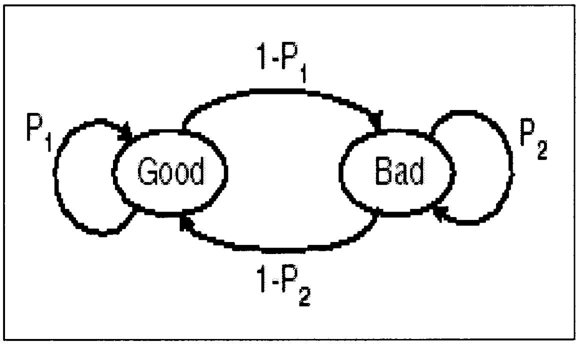


Figure 3.4 Two-state Markov chain for packet drop process.

In the two-state Markov model, a channel can be in one of two possible states, "good" or "bad." The state transition diagram is shown in Figure 3.4. The probability of dropping a packet, i.e., the probability of a packet error, is different in each state. P_G is the probability of dropping a packet while in the good state and P_B, $P_B > P_G$, is the probability of dropping a packet while in the bad state. Given a present state, a channel may transfer to the other state or stay in the present state with certain probabilities. P_1 and P_2 are the transition probabilities of staying in the good and bad states, respectively.

(II) Constrained capacity

The capacity of wireless links may be less than the capacity of the wired links used in the test bed. Constraints on available bandwidth are based on a leaky bucket token buffer model.

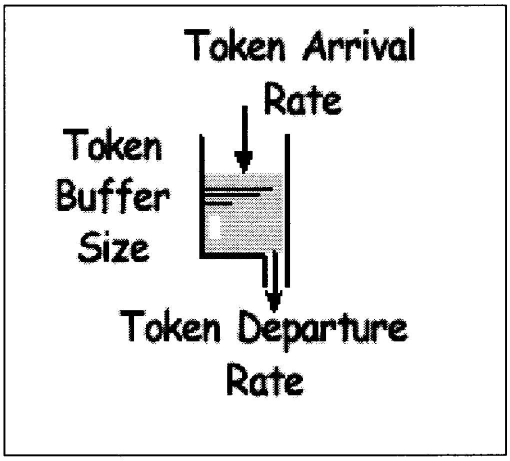


Figure 3.5 A Leaky Bucket Token Buffer Model.

In the leaky-bucket token buffer model, no packet can be sent unless there is a token in the token buffer or a new token arrives. There is an upper bound on the size of the token buffer. A token arrival rate of r tokens per second, a token buffer size of B tokens, and an allowable transmission size of μ. bytes per token are used to determine the bandwidth constraint. The equation below specifies the maximum allowable data rate, where C is the transmission rate or emulated capacity.

$$C = \mu * r \text{ bytes per second}$$

3.2.5 Limitation of Dynamic Topology Switch

- The link layer effects cannot be emulated, since the dynamic switch software did not alter MAC layer information while transmitting packets from one interface to another.
- The other issue, which this software did not take care of, was scalability. The network size was limited due to operating system limitations. Only 11 nodes could connect to the dynamic switch, thus a wireless network with only 11 nodes talking to each other.

3.2.6 The proposed test bed for collection of MIB Data in wireless networks

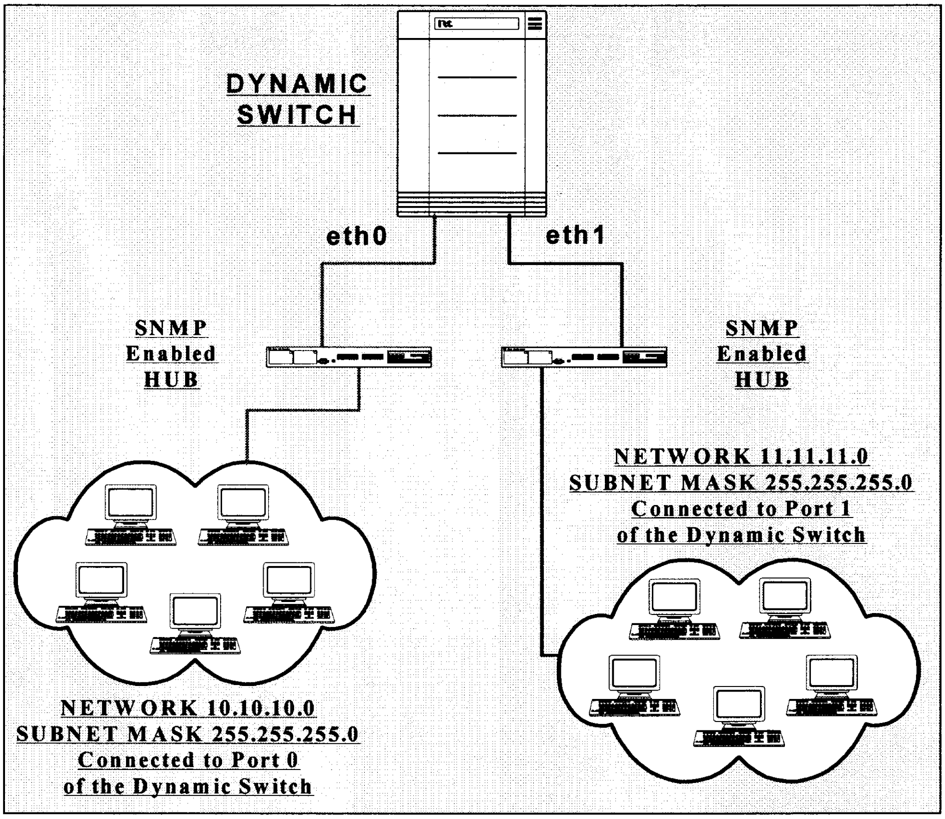


Figure 3.6 Emulation of MANET using the Dynamic Topology Switch.

Keeping in mind the scalability issue, the above network test bed was proposed. Here we would have used just two interfaces on the dynamic switch, with each port being a part of different subnets (10.10.10.0 and 11.11.11.0). A hardware switch acting as a layer 2 hub could be connected to each interface. Nodes could be attached to this hardware switch.

The same client server paradigm [4,5,6,7] could be used to simulate internet traffic (background and attack traffic). A background and fault server could be run on each subnet with their corresponding clients running on the other subnet. This way all clients on one subnet would be talking to their corresponding server on the other subnet, through the wireless channel emulated by the dynamic switch. SNMP (MIB-II) based data collection could have been done on both the servers.

This proposed test bed could have removed the scalability limitation, of the dynamic switch software, to some extent.

3.2.7 Second test bed proposed for collection of MIB Data in wireless networks

Due to the limitations faced during the use of software switch, another network test bed was considered to simulate internet traffic and collect MIB data.

This time, Virtual LAN (VLAN) technology is used to create logically separate LANs on the same physical switch. Each port of the switch is assigned to a VLAN. The communication between switches and routers take place through trunking. For two nodes on two different VLANs to communicate, the data must travel from the switch to the router and back again to the switch. In this way, the nodes are like mobile users and the switch and router act as the base stations.

For this architecture using a hardware switch, there is no limit on the number of nodes that act as wireless nodes. Moreover, the hardware switch can be cascaded to support more nodes. An example network using this test bed scheme is shown below.

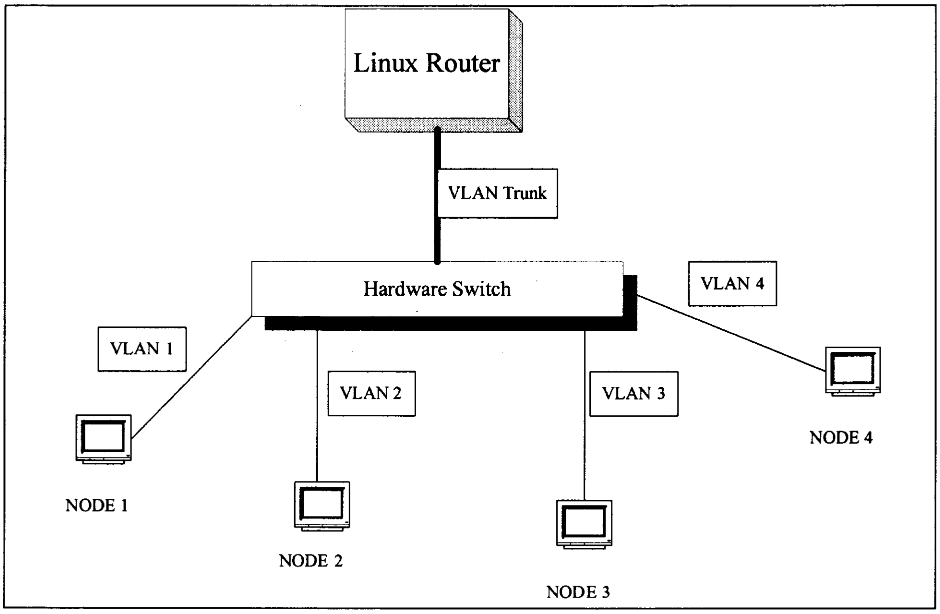


Figure 3.7 Another Test bed to emulate wireless network.

Hardware switch consists of three components:

VLAN routing, traffic control, and packet error rate. The first two components are becoming the Linux standards. We can use them directly and focus on the design of the channel simulator.

A random packet dropper function can be added to the Linux kernel of the router machine, to simulate channel simulator with packet error rate. By setting different packet drop probabilities, the router acts as a wireless channel.

To simulate Traffic control, we can use a new linux package called Linux advanced routing (*iproute2*), which contains a traffic control mechanism (*tc*). The traffic control system sends packet to filters and queues to implement flow control and bandwidth allocation.

CHAPTER 4

EXPERIMENT - DATA GENERATION AND COLLECTION

Understanding the nature of network traffic is critical in order to properly design and implement computer networks. Ethernet traffic exhibits self-similar [1,2,9] properties. Self-similarity means that an object is composed of sub-units and sub-sub-units on multiple levels that (statistically) resemble the structure of the whole object.

Self-similarity describes the phenomenon where a certain property of an object is preserved with respect to scaling in space and/or time. The background and fault (attack) traffic generated during the experiments done here followed two distributions in order to generate self-similar traffic, namely Pareto and Weibull distributions.

4.1 Procedure

All experiments and data collection were done on the wired network as shown in Figure 4.1. The network test bed was operated using the Client – Server paradigm, with the management station running in the server. The management station collected 27 traffic parameters corresponding to four MIB groups: IF, IP, TCP and UDP, for each network element monitored, i.e., 9 parameters from the Interface Group, 6 from the IP Group, 4 from the UDP Group, and 8 from the TCP Group. For example, the UDP group consisted of the udpInDatagrams, udpInNoPorts, udpInErrors and udpOutDatagrams parameters. The selection of the specific subset of the traffic parameters is based on their relevance to the network traffic and their inter-relationship, derived from domain knowledge as well as from some correlation studies.

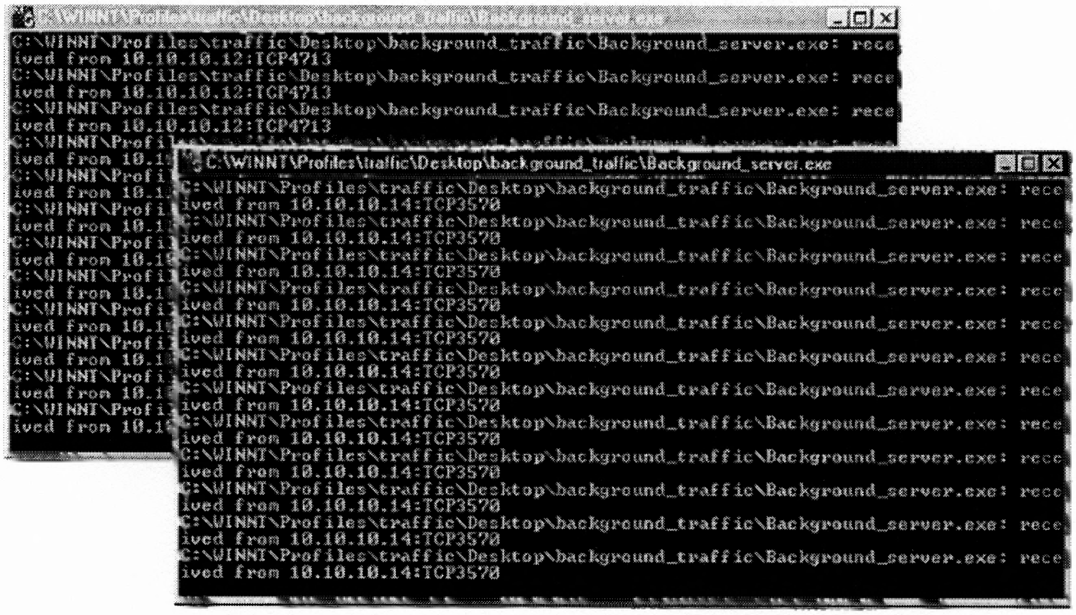
Background and fault traffic were generated with different intensities to simulate the real-time internet traffic. For this purpose, two client server programs were run, one to simulate background traffic and the other to simulate fault traffic (attack traffic). The background and fault traffic servers were ran on a victim machine in the Victim VLAN, background traffic clients were ran on the traffic subnet and a fault traffic client on the Attack VLAN.

The background traffic clients generated packets of sizes that followed the Pareto distribution. The rate at which the packets were generated followed the Weibull distribution.

The fault traffic model generated network fault traffic in a periodic pulse. The time period in seconds was given as the input.

The size of the packet could follow any of the following distribution: Exponential Distribution, Constant Distribution or Pareto Distribution. The packet rate could follow any of the following distribution: Exponential, Constant, Pareto or Weibull Distribution.

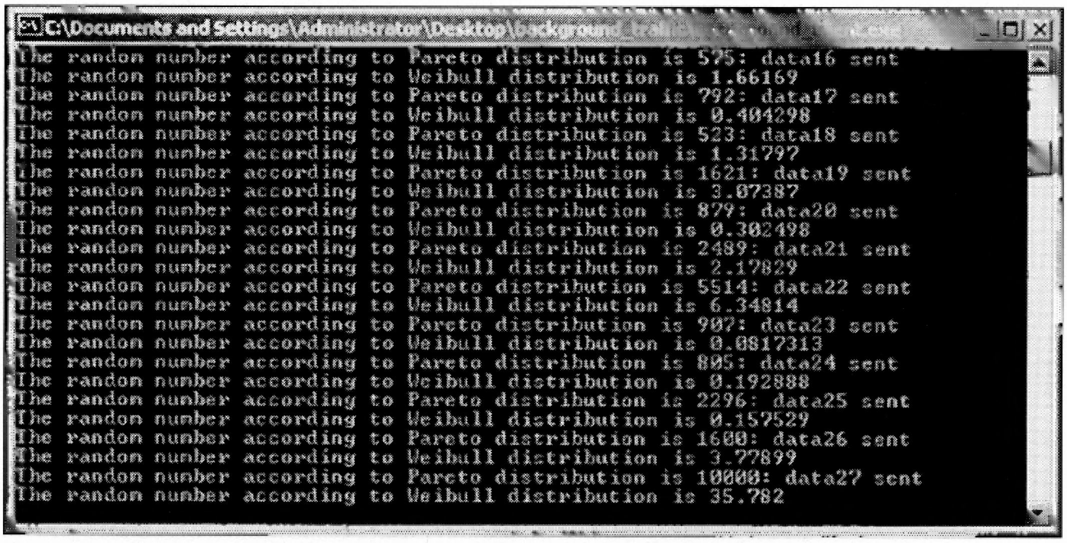
The figures on the next two pages show an example of running background traffic servers/clients and fault traffic server/client programs.


C:\WINNT\Profiles\traffic\Desktop\background_traffic\Background_server.exe
C:\WINNT\Profiles\traffic\Desktop\background_traffic\Background_server.exe

```
C:\WINNT\Profiles\traffic\Desktop\background_traffic\Background_server.exe: rece
ived from 10.10.10.12:TCP4713
C:\WINNT\Profiles\traffic\Desktop\background_traffic\Background_server.exe: rece
ived from 10.10.10.12:TCP4713
C:\WINNT\Profiles\traffic\Desktop\background_traffic\Background_server.exe: rece
ived from 10.10.10.12:TCP4713
C:\WINNT\Profi
ived from 10.1
C:\WINNT\Profi
ived from 10.1
C:\WINNT\Profi
ived from 10.1
C:\WINNT\Profi
ived from 10.1
C:\WINNT\Profi
ived from 10.1
C:\WINNT\Profi
ived from 10.1
C:\WINNT\Profi
ived from 10.1
C:\WINNT\Profi
ived from 10.1
C:\WINNT\Profi
ived from 10.1
```

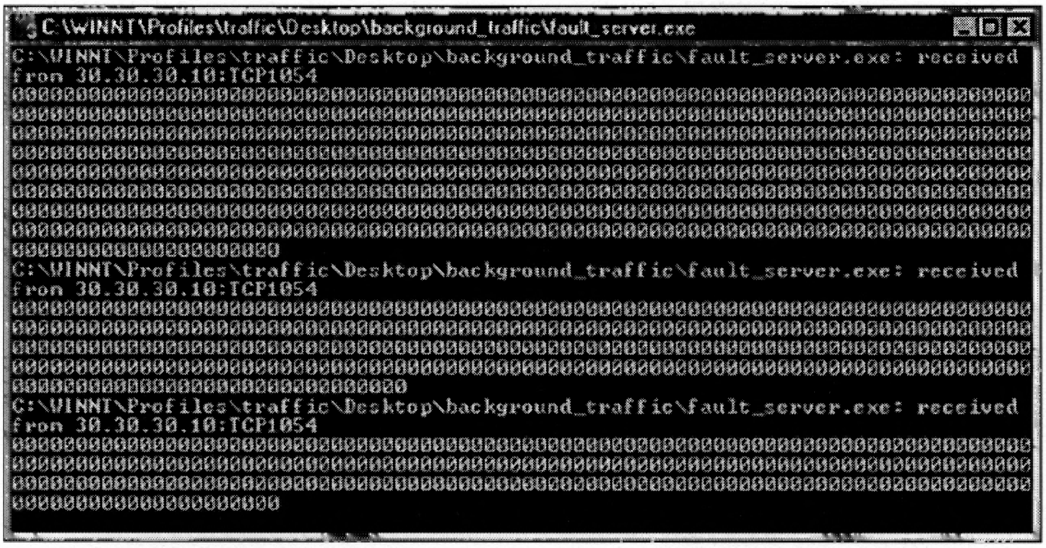
```
C:\WINNT\Profiles\traffic\Desktop\background_traffic\Background_server.exe: rece
ived from 10.10.10.14:TCP3570
C:\WINNT\Profiles\traffic\Desktop\background_traffic\Background_server.exe: rece
ived from 10.10.10.14:TCP3570
C:\WINNT\Profiles\traffic\Desktop\background_traffic\Background_server.exe: rece
ived from 10.10.10.14:TCP3570
C:\WINNT\Profiles\traffic\Desktop\background_traffic\Background_server.exe: rece
ived from 10.10.10.14:TCP3570
C:\WINNT\Profiles\traffic\Desktop\background_traffic\Background_server.exe: rece
ived from 10.10.10.14:TCP3570
C:\WINNT\Profiles\traffic\Desktop\background_traffic\Background_server.exe: rece
ived from 10.10.10.14:TCP3570
C:\WINNT\Profiles\traffic\Desktop\background_traffic\Background_server.exe: rece
ived from 10.10.10.14:TCP3570
C:\WINNT\Profiles\traffic\Desktop\background_traffic\Background_server.exe: rece
ived from 10.10.10.14:TCP3570
C:\WINNT\Profiles\traffic\Desktop\background_traffic\Background_server.exe: rece
ived from 10.10.10.14:TCP3570
C:\WINNT\Profiles\traffic\Desktop\background_traffic\Background_server.exe: rece
ived from 10.10.10.14:TCP3570
C:\WINNT\Profiles\traffic\Desktop\background_traffic\Background_server.exe: rece
ived from 10.10.10.14:TCP3570
C:\WINNT\Profiles\traffic\Desktop\background_traffic\Background_server.exe: rece
ived from 10.10.10.14:TCP3570
```

Figure 4.1 Background servers receiving packets from the corresponding clients.


C:\Documents and Settings\Administrator\Desktop\background_traf

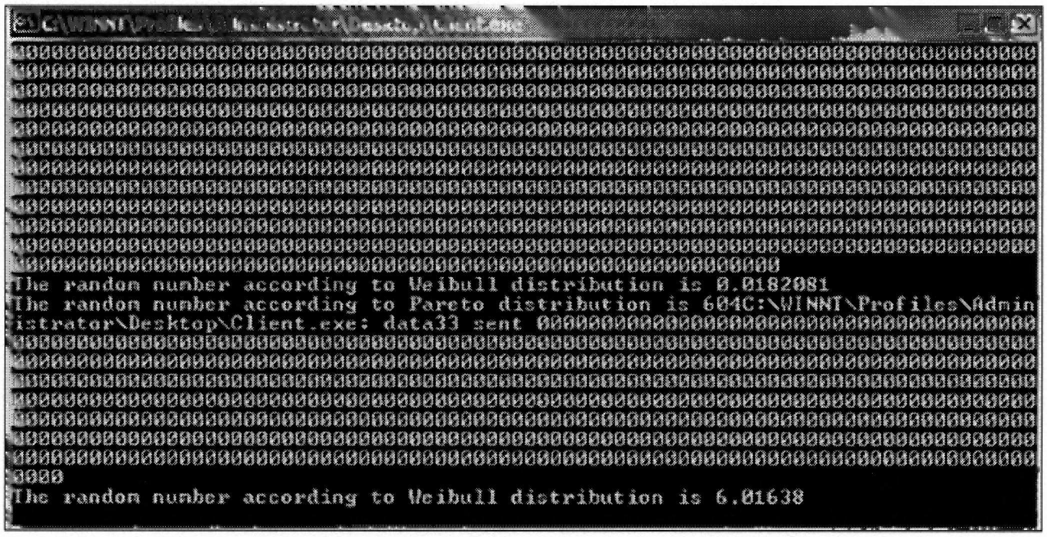
```
The random number according to Pareto distribution is 575: data16 sent
The random number according to Weibull distribution is 1.66169
The random number according to Pareto distribution is 792: data17 sent
The random number according to Weibull distribution is 0.404298
The random number according to Pareto distribution is 523: data18 sent
The random number according to Weibull distribution is 1.31797
The random number according to Pareto distribution is 1621: data19 sent
The random number according to Weibull distribution is 3.07387
The random number according to Pareto distribution is 879: data20 sent
The random number according to Weibull distribution is 0.302498
The random number according to Pareto distribution is 2489: data21 sent
The random number according to Weibull distribution is 2.17829
The random number according to Pareto distribution is 5514: data22 sent
The random number according to Weibull distribution is 6.34814
The random number according to Pareto distribution is 907: data23 sent
The random number according to Weibull distribution is 0.0817313
The random number according to Pareto distribution is 805: data24 sent
The random number according to Weibull distribution is 0.192888
The random number according to Pareto distribution is 2296: data25 sent
The random number according to Weibull distribution is 0.157529
The random number according to Pareto distribution is 1600: data26 sent
The random number according to Weibull distribution is 3.77899
The random number according to Pareto distribution is 10000: data27 sent
The random number according to Weibull distribution is 35.782
```

Figure 4.2 Background clients sending out packets to the corresponding servers.



```
C:\WINNT\Profiles\traffic\Desktop\background_traffic\fault_server.exe: received
from 30.30.30.10:TCP1054
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000
C:\WINNT\Profiles\traffic\Desktop\background_traffic\fault_server.exe: received
from 30.30.30.10:TCP1054
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000
C:\WINNT\Profiles\traffic\Desktop\background_traffic\fault_server.exe: received
from 30.30.30.10:TCP1054
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000
```

Figure 4.3 Fault traffic server program receiving packets.



```
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
The random number according to Weibull distribution is 0.0182081
The random number according to Pareto distribution is 604C:\WINNT\Profiles\Admin
istrator\Desktop\Client.exe: data33 sent 000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
0000
The random number according to Weibull distribution is 6.01638
```

Figure 4.4 Fault traffic client sending out packets to Fault traffic server.

In conducting the attack, one client in the “Attack VLAN” sends the attack packets to the “System/Victim VLAN”. The attack packet length is 100 bytes with a constant distribution, while the attack packet rate will vary according to the different attack intensity scenarios.

For each simulation scenario, network traffic was collected over a duration of 10 hours. The data were recorded periodically, with a period of one second.

4.2 The Traffic Monitor

The data collection throughout this research was carried out using a commercially used IpSwitch, Inc. network management tool, called *WhatsUp Gold* [10].

WhatsUp Gold, a graphical network mapping, monitoring and notification solution that helps keeping the network up and running, is a very useful tool to monitor a wide range of devices, applications, and services for network problems. With WhatsUp Gold one can easily filter through events and locate the log data for any type of network event by searching on event type, IP address or device name. This saves time, provides more flexibility, and also gives users more control since they can pinpoint information without having to sort through a flood of event data.

WhatsUp Gold automatically detects the devices and hosts in a network and then creates a network map. It polls all the systems in the network with a polling frequency that can be set by the user. With the auto discovery wizard, it creates an accurate representation of the network based on information contained in the host computer or on the network. Auto discovery was used to create a map of the test bed (figure 3.1) as shown below.

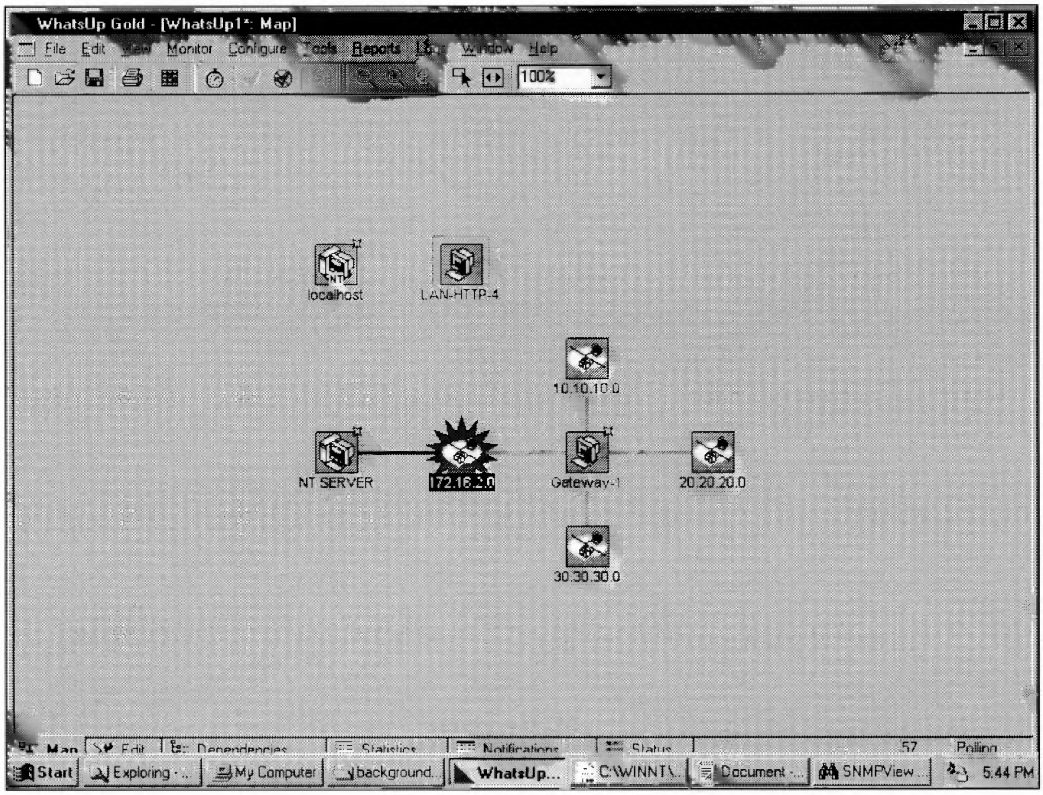

WhatsUp Gold - [WhatsUp1*: Map]
File Edit View Monitor Configure Tools Reports Log Window Help
100%
localhost
LAN-HTTP-4
10.10.10.0
NT SERVER
172.16.2.0
Gateway-1
20.20.20.0
30.30.30.0
Map Edit Dependencies Statistics Notifications Status
57 Polling
Start Exploring - ... My Computer background. WhatsUp... C:\WINNT\... Document - ... SNMPView ... 5:44 PM

Figure 4.5 Network map as detected by WhatsUp Gold.

Links between SNMP enabled devices provided color-coded indication of their up or down status. Links, which were down and attribute the outage to a specific device interface, could be seen using this tool. Devices with multiple interfaces are fully supported in network mapping, monitoring and alert notification - allowing accurate depiction of the network topology and to pinpoint network problems down to a device's interface, either physical or virtual.

In this research, this tool was used to monitor SNMP enabled devices for acceptable ranges of specific MIB variables. The SNMPView feature of this tool displayed the status of interfaces on a device and let us view MIB values. MIB traffic data could also be graphed to show throughput in real-time.

The SNMP Viewer displayed an icon for each interface on the device .A Typical example is as shown below.

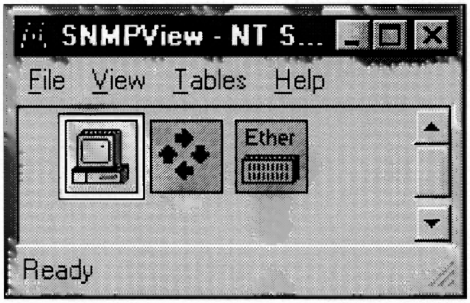
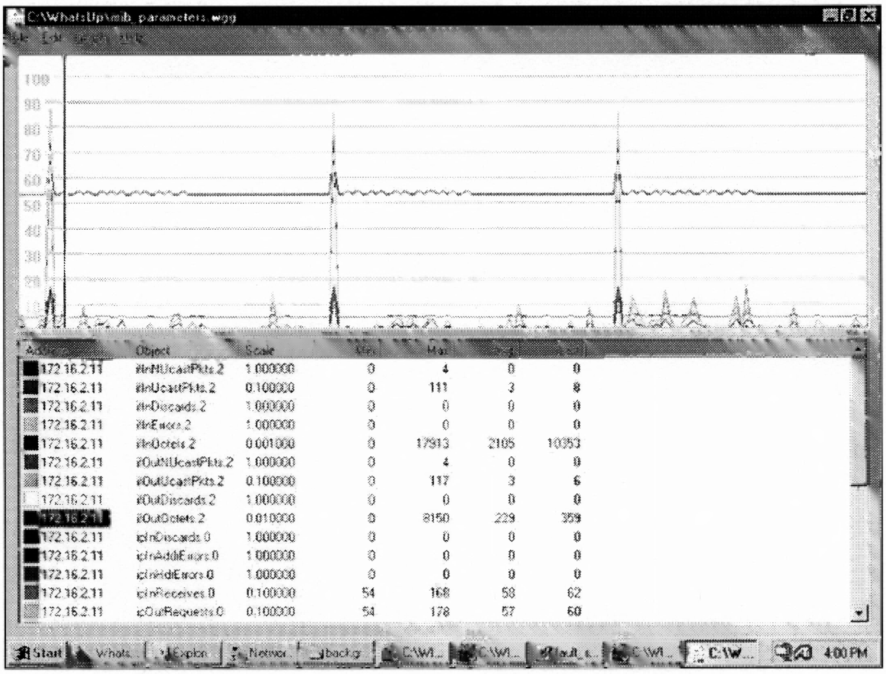


Figure 4.6 SNMP Viewer.

The first icon from the left gave the MIB parameters for the System Group, the second one gave the loop-back interface information, and the third one gave the MIB parameters associated to the actual ethernet interface on the computer.

The following snapshot shows the WhatsUp Gold monitor the 27 MIB parameters monitored during this research.



Address	Object	Scale	Min	Max	Avg	Last
172.16.2.11	ifInNUcastPkts.2	1.000000	0	4	0	0
172.16.2.11	ifInUcastPkts.2	0.100000	0	111	3	8
172.16.2.11	ifInDiscards.2	1.000000	0	0	0	0
172.16.2.11	ifInErrors.2	1.000000	0	0	0	0
172.16.2.11	ifInOctets.2	0.001000	0	17913	2105	10353
172.16.2.11	ifOutNUcastPkts.2	1.000000	0	4	0	0
172.16.2.11	ifOutUcastPkts.2	0.100000	0	117	3	6
172.16.2.11	ifOutDiscards.2	1.000000	0	0	0	0
172.16.2.11	ifOutOctets.2	0.010000	0	8150	229	359
172.16.2.11	ipInDiscards.0	1.000000	0	0	0	0
172.16.2.11	ipInAddrErrors.0	1.000000	0	0	0	0
172.16.2.11	ipInHdrErrors.0	1.000000	0	0	0	0
172.16.2.11	ipInReceives.0	0.100000	54	168	58	62
172.16.2.11	ipOutRequests.0	0.100000	54	178	57	60

Figure 4.7 Traffic monitoring using WhatsUp Gold.

CONCLUSION

After monitoring background and fault traffic on wired network, it is proposed that similar collection of MIB data can be done on a wireless network as well. This paper has proposed two test beds to emulate a wireless model in a wired environment and facilitate similar collection of MIB data. The data collected using both wired and wireless network test beds may be used for experimentation purposes for the development of the fault detection system, which would be a part of the final anomaly network intrusion detection system.

REFERENCES

[1] Kihong Park, Gitae Kim, and Mark E. Crovella, On the Effect of Traffic Self-Similarity on Network Performance. *In Proceedings of SPIE International Conference on Performance and Control of Network Systems, November, 1997.*

[2] Walter Willinger, Feldmann, Gilbert A.C., and Kurtz T.G, The Changing Nature of Network Traffic: Scaling Phenomena, *ACM Computer Communication Review, vol. 28, pp. 5-29, Apr. 1998.*

[3] Subhasish Ghosh, Compiling and Installing a Linux Kernel, *published in Issue 68 of Linux Gazette, July 2001.*

[4] Documentation on Linux Networking available on
http://www.redhat.com/docs/manuals/linux/ (4 May 2003)

[5] Rob Tougher, Linux Socket Programming In C++, *published in Issue 74 of Linux Gazette, January 2002.*

[6] Gent Hito, A chronicle of how the /n software team ported the IP*Works! Internet Toolkit from Delphi to Kylix. *Article ID: 26793, published on September 28, 2001.*

[7] Example code for Linux sockets available at
http://alf.fei.tuke.sk/pai/examples.html (7 May 2003)

[8] Tao Lin, Scott F. Midkiff and Jahng S. Park, *Local Computer Networks, 2002. Proceedings. LCN 2002. 27th Annual IEEE Conference on , 2002 Page(s): 791 – 798.*

[9] Will Leland, Murad Taqqu, Walter Willinger, and Daniel Wilson, On the Self-Similar Nature of Ethernet Traffic (Extended Version), *IEEE/ACM Transactions on Networking, Vol. 2, No. 1, pp. 1-15, February 1994.*

[10] Documentation on WhatsUp Gold – Network Monitoring Tool, available at
http://www.ipswitch.com/Support/whatsup/guide/v700/WUG7TOC.html

[11] Tammy Fox ,Network Configuration Using the Command Line, available at
http://www.linuxheadquarters.com/howto/networking/networkconfig.shtml
(30 April)

[12] Proactive and Adaptive Detection of Network/Service Anomalies, available at
http://www.ece.rice.edu/~zpjian/ELEC531/index.html (10 May 2003)

[13] Cynthia S. Hood, Intelligent Agents for Proactive Fault Detection, Illinois Institute of Technology.

[14] Cynthia S. Hood, Proactive Network Fault Detection, Department of Computer Science and Applied Math, Illinois Institute of Technology, Chicago, IL 60616.

[15] Marina Thottan, Properties of Network Faults, Bell Laboratories, Holmdel, NJ.

[16] 802.1Q VLAN implementation for Linux, available at http://www.candelatech.com/~greear/vlan.html (15 May 2003)

[17] Management Information Base for Network Management of TCP/IP-based internets: MIB-II (RFC 1213), available at http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1213.html (21 Jan 2003)

[18] Simple Network Management protocol, available at http://www.synapse.de/ban/HTML/P_TCP_IP/Eng/P_tcp103.html (15 May 2003)

[19] Overview of SNM and MIB, available at http://www.et.put.poznan.pl/snmp/intro/ioverviv4.html (15 May 2003)