CAMPUS NETWORK TOPOLOGY DISCOVERY AND DISTRIBUTED FIREWALL POLICY GENERATION

A Thesis Submitted to
the Graduate School of Engineering and Sciences of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in Computer Engineering

by
Ezgi ÇALIŞKAN

March 2011
İZMİR

We approve the thesis of **Ezgi ÇALIŞKAN**

Assist. Prof. Dr. Tuğkan TUĞLULAR
Supervisor

Assist. Prof. Dr. Tolga AYAV
Committee Member

Assist. Prof. Dr. Enis KARAARSLAN
Committee Member

4 March 2011

Prof. Dr. Sıtkı AYTAÇ
Head of the Department of
Computer Engineering

Prof. Dr. Durmuş Ali DEMİR
Dean of the Graduate School of
Engineering and Sciences

ACKNOWLEDGMENTS

I thank S. Murat Özkan. He is one of the people who developed FERGI for the first time with me. I thank Assist. Prof. Dr. Tuğkan Tuğlular for his supports and encouragements. I thank Mesut Güngör for his support and patience. I thank Dr. Ebru Melek Koç for her help on writing the thesis.

ABSTRACT

CAMPUS NETWORK TOPOLOGY DISCOVERY AND DISTRIBUTED FIREWALL POLICY GENERATION

The change in technology of network components has enabled more complex and dynamic computer networks to occur. At present, most network components can easily be attached to or removed from computer networks. This situation causes the static prevention techniques to be inadequate. In static prevention, any situation which is different than expected ones occurs, the default rule is taken granted for it. Detecting unpredictable situations and finding out solutions for them takes time.

There are some network systems, which control network parameters dynamically, such as intrusion detection systems integrated firewalls. However, even if these systems control traffic parameters, they can only alert when the parameter values are not in the given range. They may not be successful to determine well-designed attacks or even if the system determines the attack, it takes time to interfere.

Instead of static approaches, a dynamic network security system, which is compatible with dynamic network topology and can update the security issues according to changes in network, is needed. To achieve this dynamic nature, the network must be monitored. Then controlling and managing new components could be easier and more secure.

New security rules must be created for the newly attached network components or security rules must be removed for removed network components. In this thesis, an approach to monitor a campus area network and dynamically update firewall rules according to monitoring results is proposed. The implemented approach is validated through a case study.

ÖZET

KAMPÜS AĞ TOPOLOJİSİNİN BULUNMASI VE DAĞITIK GÜVENLİK DUVARI POLİTİKASI OLUŞTURULMASI

Ağ bileşenleri teknolojilerinin değişmesi daha karmaşık ve dinamik ağların oluşmasını sağladı. Günümüzde çoğu ağ bileşeni çok kolaylıkla bilgisayar ağlarına eklenebiliyor ve ağdan çıkartılabiliyor. Bu durum sabit korunma yöntemlerini etkisiz bırakıyor. Sabit korunma yöntemlerinde, beklenenin dışında bir durum görüldüğünde, ön tanımlı kural geçerli sayılıyor. Bu beklenmeyen durumun fark edilmesi ve bir çözüm üretilmesi de zaman alıyor.

IDS (Intrusion Detection Systems) gibi ağı dinamik olarak kontrol eden sistemler de var. Ancak, bu sistemler ağ trafiğini dinamik olarak kontrol etseler bile sadece belli değer aralığının dışındaki durumlarda uyarı verirler. İyi tasarlanmış ağ ataklarında başarılı olamayabilirler ya da atağı tespit etse bile müdahale etek zaman alır.

Tüm bu sabit yaklaşımların yerine, ağdaki dinamik değişikliklere duyarlı olan ve güvenlik unsurlarını bu değişikliklere göre düzenleyen bir sisteme ihtiyaç var. Bu dinamik yapıyı sağlayabilmek için, ağ sürekli izlenmeli. Bu sayede, ağa yeni katılan ağ bileşenlerinin kontrol edilmesi ve yönetilmesi de daha kolay ve daha güvenli olacaktır.

Yeni güvenlik kuralları ağa yeni katılan ağ bileşenleri için oluşturulmalı. Bu tezde, bir kampus ağını izleyen ve ateş duvarı kurallarını bu bilgiler doğrultusunda dinamik olarak güncelleyen bir yaklaşım hedeflendi. Hayata geçirilen yaklaşımın doğrulama çalışmaları da yapıldı.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

The computer networks have started to be more dynamic and complex with the change of computer network technology. Until recently, hosts could be joined to network only through cables. However, currently any computer can do it using wireless cards without any need of cables. With the ability that networks gain, the currently-used static prevention techniques becoming less and less adequate. For instance, a packet filter firewall, which looks at each packet entering or leaving the network and accepts or rejects it based on user-defined rules, uses expected scenarios and if any situation, which is different from expected ones occurs, the default rule will be taken granted for it. Detecting this unexpected new situation, finding out a solution for it and then implementing the solution takes time and in the meantime undesirable situations may occur. For instance, assume that an IP address was given to a host. Then the host moved to an access point. If one of the hosts sharing the access point's computer network exploits the traffic, the firewall blocks all the traffic of the access point's traffic. This is an undesirable result.

There are some systems which control network parameters dynamically, such as intrusion detection system (IDS) integrated firewalls. Even if these systems control traffic parameters, they can only alert when the parameter values are in the given range. They cannot successfully determine well-designed attacks and even if the system determines the attack, it may take time. For instance, there is an IDS system which has a normal value of TCP in a period of a month; anyone wants to do denial-of-service attack to any server in computer network by using a method such as unsolicited resetting of TCP sessions; IDS system will not alert unless TCP traffic is over the normal value. Even if the IDS system alerts, the attack has started at time t1 and the IDS system realizes it at time t2; this means there is a time worth of t2-t1.

The goal of this thesis is to provide a more secure network system by proposing an approach and then implementing it in a tool, which generates rules for firewalls parallel to the dynamic changes in computer network. Current firewalls cannot keep up with the dynamic structure of the computer networks. They run only according to administrators' predictions. With the approach and the tool introduced, the dependency to network security to administrators' static rules decreases so it becomes more flexible and the time spent by recognizing the changes in computer network then changing firewall rules will decrease. The tool aims to provide real time prevention and to achieve this, it performs periodic traces to the network it protects. The scope of the thesis is to enforce network security policy in a campus networks protected by an edge firewall and other firewalls between edge firewall and host computers by rule generation with respect to the network topology.

The tool developed and named as FERGI (Firewall Rule Generation Interface) works integrated with firewalls. It generates rules according to the changes in the computer network. To recognize the network state, hop-by-hop strategy is used: firstly by trace routing to any given web site the hops which mean our gateways are found. Then using the gateway IP address, the class of gateway is calculated such as A, B, C. The class information of gateway gives us the subnet mask value. All the possible IP addresses in this range are pinged. After getting alive IP addresses, the active ports of each host are found by using port scanning. When the recognizing topology step is achieved, then rule generation starts. To generate rules, global and local policies are used. The aim in using two policy approaches is to generate more specific rules for the sub-networks. The last step is FERGI modules communicating among each other. A FERGI module can find its parent FERGI and inform it about its findings. FERGI modules are located at the gateways like firewalls.

The thesis is structured as follows: In Chapter 2 the methods and strategies which are used to discover network topology will be mentioned. In Chapter 3, the network topology discovery approach of the thesis will be introduced. In Chapter 4, the firewall rule generation approach of the thesis will be mentioned. In Chapter 4, the implementation details and product screen shots of the thesis will be introduced. Chapter 6 is the chapter in which the validation tests and the test results are mentioned. Chapter 7 is the conclusion chapter of the thesis.

CHAPTER 2

INTRODUCTION TO TOPOLOGY DISCOVERY

Topology discovery means discovering network components and the relation among them. There are two types of methods to discover network topology: passive monitoring and active probing. By using methods, three types of topology discovery strategies are developed: benchmark, hop by hop and hybrid. In this chapter, these methods and strategies will be introduced.

2.1. Methods

Network components are discovered according to the packets which they generate. To discover the network components two methods are used: passive monitoring and active probing.

2.1.1. Passive Monitoring

Passive monitoring is a method which is used to extract data only by capturing traffic from network. There are advantages to use the method. One of them is that no additional traffic is introduced into the network. Another advantage of passive monitoring is that it measures performance for most popular destination hosts. The hosts which interact often with local hosts generate more traffic. As a result, there are more network measurements [1].

Although the non-intrusive nature of passive monitoring is very appealing, it has disadvantages. One of the disadvantages is that passive monitoring can only measure regions of the Internet that application traffic has previously traversed. The second disadvantage is that passive monitoring may be forced to recollect most of its distance information when internet topology changes. Finally, passive monitoring typically requires measurement or snooping of network traffic, which may raise privacy and security concerns [2].

2.1.2. Active Probing

Active probing is a method which is used to extract data from response packets of generated packets. For instance, to detect an IP address is alive or not ICMP packets are sent to the IP address. Then response packets are listened. This is called as active probing [3].

Active probing has two layouts; one of them is the single-ended layout. The single ended layout enables the active probe tester to control the end points [3]. It is as shown in Figure 2.1.

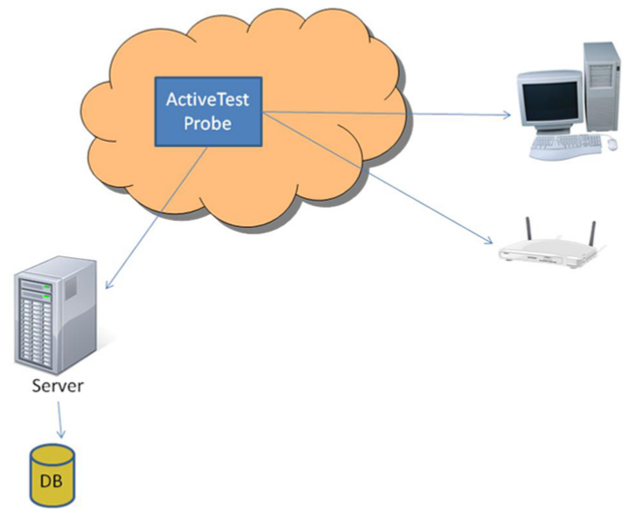


Figure 2.1. Single Ended Layout

The second layout is the Probe-to-Probe Layout. The probe to probe layout enables the active probe tester to control both end points and other active probe testers. It is as shown in Figure 2.2:

Probing technology has many advantages. First of all, it does not require extra instrumentation and works with any server that takes user transactions. The second advantage is that it is very flexible; a probe station can be placed in any location with network access and can target multiple components. However, active probing has disadvantages. When probes are used, additional network and server load, the need to collect, store and analyse probe results may occur. This imposes costs. It is important to control these costs in order to use probing effectively [3].

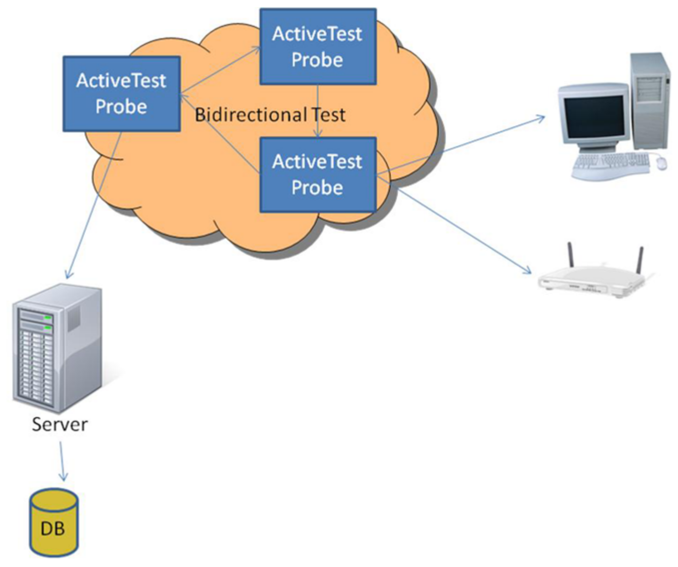


Figure 2.2. Probe to Probe Layout

2.2. Strategies

There are many studies on topology discovery topic. Lowekamp, O'Hallaron and Gross classified them into three types of strategies: benchmark strategy, hop by hop strategy and hybrid strategy [4]. In the section below, these strategies will be mentioned.

2.2.1. Benchmark Strategy

The most common strategy to determine WAN topology is benchmark strategy [2]. Benchmark strategy means using benchmarks measured across the topology to discover the network. This strategy aims to calculate a map which offers a global view of a computer network [2]. In the subsections below, the project using benchmark strategy will be introduced.

2.2.1.1. Topology-d

Topology-d uses RTT and bandwidth values to discover the topology. A group of web servers are chosen as clients. Topology-d software is installed on one of the clients.

Each client periodically sends time stamped UDP packet to each member in the group to estimate RTT (Round Trip Time) between itself and all the other machines in the group. For bandwidth estimates, each machine sends a block of data of which block size is at least 32 KB using. Then available bandwidth is computed as

$$Bandwidth = BytesSent \diagup (Time_{LastByte} - Time_{FirstByte}) \tag{2.1}$$

where the two timestamps are taken by the destination machine.

A group member designated as the master collects the estimates. These estimates are reported by group members into a cost matrix for the group which the master uses to compute the group's logical topology. Topology-d then generates the logical topology by invoking a topology generator program. The topology generator first computes a minimum cost spanning tree connecting all the nodes. And then, for each node whose degree is less than the required connectivity, it adds the current cheapest edge until node degree is the same as the required connectivity. The master periodically sends the current logical topology to all the machines in the group [5].

2.2.1.2. IDMaps

IDMaps project is developed to generate a distance map which shows the network components and the relations among them. A distance map consists of two parts: Tracer to Tracer VLs (Virtual Links) and Tracer to AP VLs. Each Tracer advertises the VLs it traces [2].

Analysing the relation network components among others is very hard, however if the network components are grouped, it is easier to detect the relations.his assumption is taken as the basic idea to develop IDMaps project. In the project, the whole network is divided into groups called Address Prefixes (APs) according to ISPN subnet values. There is a Tracer which is responsible for each address prefix. Each Tracer traces the Address Prefix which it is responsible for by using traceroute in a period of time. The operation gives the Tracer to AP Virtual Links (VLs). Tracers trace each other to calculate Tracer to Tracer VLs.

There are also Clients on which IDMaps software is installed. They are located between Tracers and all the VLs are calculated there. Then the results are sent to Tracers. Client has a full list of Tracers and their locations. The Tracer to Tracer part of the distance map is computed either by assuming a full-mesh among all Tracers or by executing the original t-spanner algorithm. To calculate Tracer to AP VLs, IDMaps project uses three different algorithms to build up distance maps: Waxman model, Tiers model and a model based on AS-connectivity [2]. The Waxman model, which is introduced by Waxman in 1988 as the random network topology generation, is a geographic model for the growth of a computer network. In this model the nodes of the network are uniformly distributed into the plane and edges are added according to probabilities which depend on the distances between the nodes [6]. The Tiers model is defined as the internetwork as a series of smaller networks and concentrate upon the relationships between these smaller networks, and the relationships between the yet-smaller networks which make up each level of a network. This follows the concept of layers in a network where each layer is built upon a number of instances of the previous layer [7].

IDMaps Client has a full list of Tracers and their locations. The Tracer to Tracer part of the distance map is computed either assuming a full-mesh among all Tracers or by executing the original t-spanner algorithm [2].

A t-spanner a graph is a sub graph where the distance between any pair of nodes is at most t times larger than the distance in the original. Formally, the t-spanner algorithm can be shown as:

1. Sort E by cost c in non-decreasing order
2. $G^{II} \leftarrow$ (V,E^{I}), $E^{I} \leftarrow \emptyset$
3. For each edge (u,v) in E do
4. If(t $*$ c((u,v))$\leq d^{I}_{G}(u, v)$)
5. $E^{I} \leftarrow$ (u,v)$\cup$ E^{I}

2.2.1.3. Theilmann and Rothermel Algorithm

Theilmann and Rothermel algorithm aims to generate a network map which is based on network components and the distances among them. The approach of the study relies on mServers which are the measurement servers. Network components are assigned to their most closely connected mServers then mServers measure the distances between itself and the network components. mServers also measure the distances between each other. So, to estimate the distances between two hosts, they use the distance between their two assigned closest mServers. The measurement approach is shown in Figure 2.3. As seen in the figure, the distances between hosts 1 and 2 and between hosts 2 and 3 are both estimated by the distance between mServers A and B. However, according to the scenario sketched in Figure 2.4, the distance between mServers A and C need not be measured if it is not known that C is close to B and B is far from A. To solve the problem, the mServers are clustered in a hierarchical manner which achieves a decomposition into regions in which each region is further refined into subregions. For each region/cluster selected a representative mServer is chosen. Then, the closest assignment for simple hosts can be done hierarchically by measuring the distance of the respective host to each representative of a top-level cluster. The cluster with the closest representative is selected and the process is continued for its sub-clusters [8].

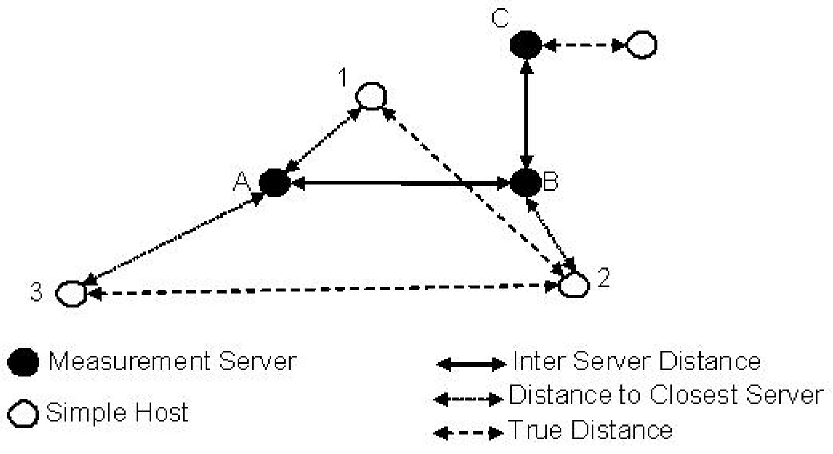


Figure 2.3. Distance estimation with measurement servers

2.2.2. Hop by Hop Strategy

It is the strategy which is used to map the Internet by sending hop-limited packets from a single location or a few locations in the computer network. Any client located in specific places sends traceroute packets to destinations which are derived from some calculations or databases such as DNS and routing tables. In subsections below, the projects using hop by hop strategy will be mentioned.

2.2.2.1. Mercator

Mercator is designed to infer a router level map. Map means a graph whose nodes represent routers in the Internet and whose links represent adjacencies between routers. Two routers are adjacent if one is exactly one IP-level hop away from the other [9].

Mercator uses hop limited probing with a heuristic called informed random address probing. According to the heuristic, ping-record route packets are sent to a uniformly randomly selected destination by increasing the hop count of the packets. It is aimed to detect the loops between routers with the increase in hop count. The response of the packets gives the hop points between the source and destination resources.

Mercator also uses another heuristic called path probing. According to this heuristic, Mercator repeatedly selects a prefix from within its population, and probes the path to an address selected uniformly from within that prefix. Like traceroute, Mercator sends UDP packets to the address with successively increasing TTL. To minimize network traffic, the path probe is self-clocking. This means that the next UDP packet is not sent until a response of the previous one has been received.

If any IP address which has not been discovered yet is detected, a process called *alias resolution* begins. Alias resolution refers finding out all the interfaces of a resource. A kind of packet is sent to the destination address, then it forces the destination resource to send unicast response packets from all the interfaces of the resource [9].

2.2.3. Hybrid Strategy

Hybrid strategy is the combination of both benchmark strategy and hop-by-hop strategy. According to this strategy, protocols are combined with each other to discover the topology. In this section, the studies on hybrid strategy will be introduced.

2.2.3.1. Study of Schwartz et al.

Schwartz et al. investigated how the protocols can be used for discovering computer network topology and then they developed a tool. In the study, EtherHostProbe is used as the first protocol to sequence through a range of machines on a local network segment to discover the Ethernet addresses of each node on the network and probes the ARP cache to determine the Internet address associated with each of these hosts. Then

Domain Naming System lookup is used to determine the host name associated with each node [10].

The second protocol set uses RIPQUERY to probe the network with RIP request packets on a local network segment to determine the gateway topology of the network. The third protocol set uses broadcast "ping" to discover the hosts on a remote network segment. However, the *ping* packets are modified to use a sequentially incremented TTL(Time To Live) mechanism so that a broadcast directed at a remote network will reach that network with TTL = 1 [10]. The block diagram of the system is shown in Figure 2.4.

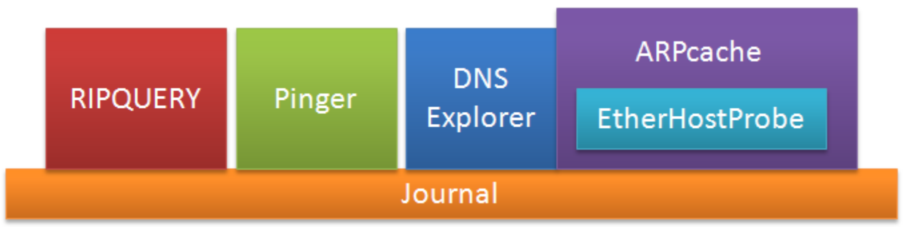


Figure 2.4. Modular Decomposition and Control Flow of System

How the project tool works is shown in Figure 2.5. The query mechanism presently consists of two major executable components, called Query and Rev_Remote_Query. Rev_Remote_Query is a server that runs continuously on each host running a network discovery suite. This component accepts TCP connections from clients and transfers the data for a particular type of discovered resource information over these connections. In the current prototype, only network and host information are discovered. Query is a stand-alone executable invoked by the interface module, as illustrated in arc 1 of Figure 2.5. Query has four command line arguments: the name of the host to query, the type of resource, the predicate query, and the name of a file to which the results of the query should be written. If the host to query is not the local host and the data is not available in the local cache, a TCP connection is made with Rev_Remote_Query running on the remote host, as illustrated from both perspectives in the arcs labelled 3 of Figure 2.5. On the remote host, Rev_Remote_Query retrieves the needed data file, as illustrated in arc 3a of Figure 2.5. The remote data file is then transferred across the connection and saved on the local machine in the cache directory under the name RemoteHostName_ResourceType, as illustrated in arc 2 of Figure 2.5. If the host to query is the local host, the local cache file is used to satisfy the query, and the TCP connection/file transfer sequence is not necessary [10].

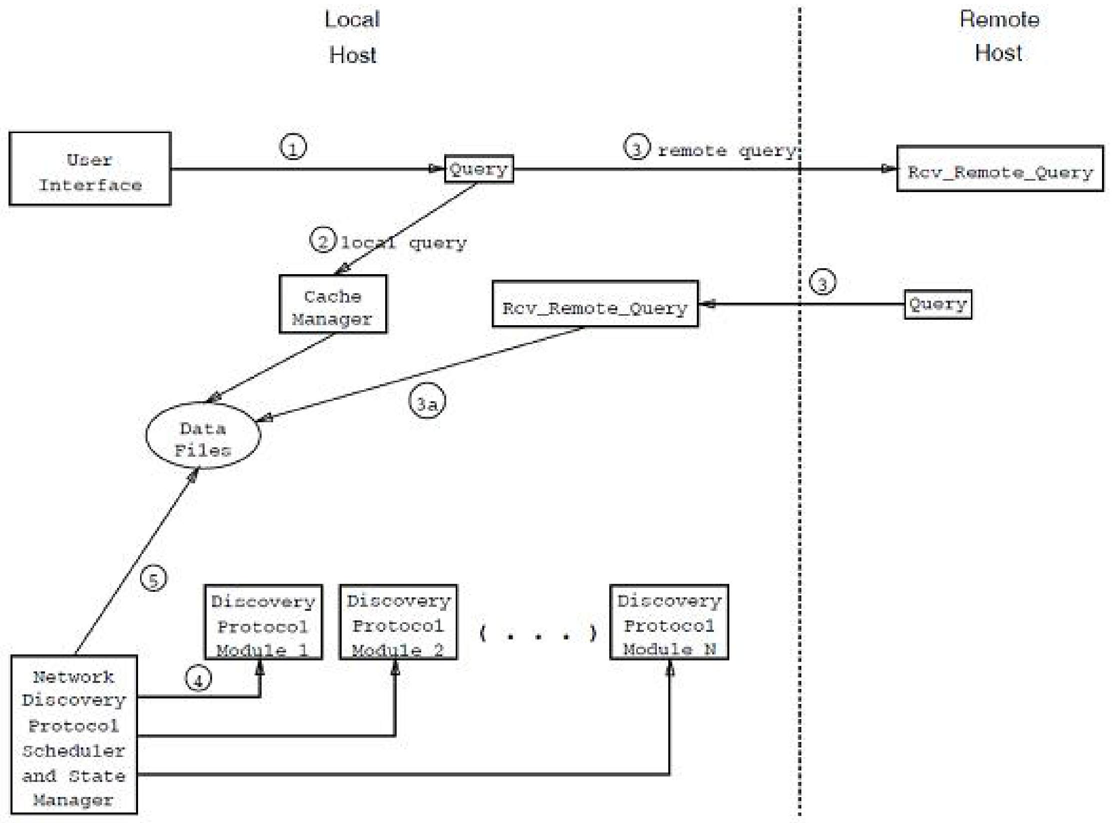


Figure 2.5. Modular Decomposition and Control Flow of System

2.2.3.2. Fremont

Fremont tool is developed to key network characteristics, such as hosts, gateways and topology. The Fremont system uses a set of modules to discover information based on different protocols and information sources, rather than a single network management protocol [11]. The general design is shown in Figure 2.6.

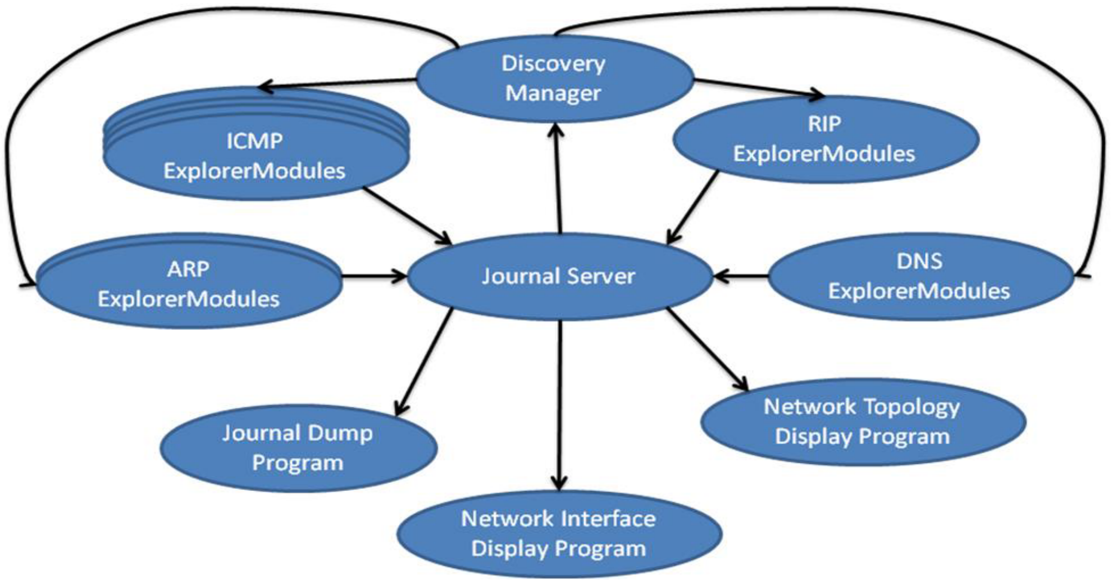


Figure 2.6. Fremont System Design

The Fremont system stores discovered information in a central repository called *Journal*. This Journal is managed by the *Journal Server*, which manages the requests for Journal. The *Discovery Manager* compares information discovered from the various Explorer Modules to determine the network.

The Fremont system is based on the *Explorer Modules*, each of which uses a network protocol or information source to discover the topology. Some of the Explorer Modules does active probing to the network while others generate no network traffic, and instead quietly observe the network activity around them.

Fremont has two Explorer Modules that use the Address Resolution Protocol to discover and record the mappings. Fremont's *ARPwatch Explorer Module* only monitors ARP message exchanges and *EtherHostProbe Explorer Module* sends an IP packet to the UDP Echo port of each host in a range of addresses.

There are four *ICMP Explorer Modules*. The first two of them are based on ICMP ECHO Request and ICMP ECHO Reply messages for identifying operational interfaces attached. The first Explorer Module is based on sequential pings. The second ICMP Explorer Module sends an ICMP Echo Request to the broadcast address of the subnet being probed. The third ICMP Explorer Module uses ICMP mask request/reply messages to determine the subnet mask of a network interface. Finally, the fourth ICMP Explorer Module uses a technique similar to Traceroute to identify gateways and the subnets to which those gateways are connected.

The *RIP Explorer Module* monitors RIP advertisements on shared subnets, building a list of hosts, subnets, and networks as they are seen in the advertisements.

The *DNS Explorer Module* searches the appropriate sub-tree for all addresses in a specified network to discover network topology by using the “nslookup” program [11]. A block diagram of Fremont implementation is shown in Figure 2.7

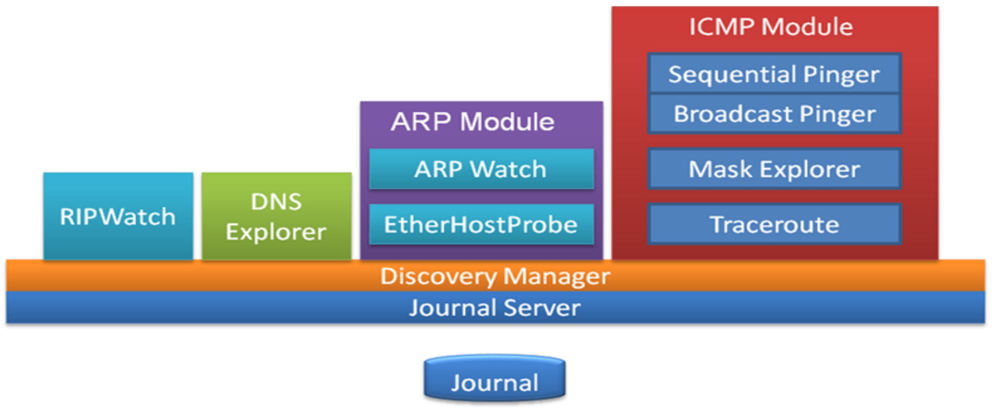


Figure 2.7. Fremont Implemetation

2.2.3.3. REMOS

REMOS is a project which provides network state to applications with a query-based interface. It is designed for experiments with the coupling of network aware applications and network architectures [4]. Network awareness is the ability of knowing detailed information about network components which are controlled [12].

REMOS Project has two main modules inside: a Collector and a Modeller. The Collector retrieves information which contains both static topology and dynamic bandwidth information from routers using SNMP. Also the Collector assumes per-hop delay for latency. The modeller calculates logical topology according to the information which the Collector provides and satisfies the application requests based on the logical topology [4]. The implementation of the project is shown in Figure 2.8.

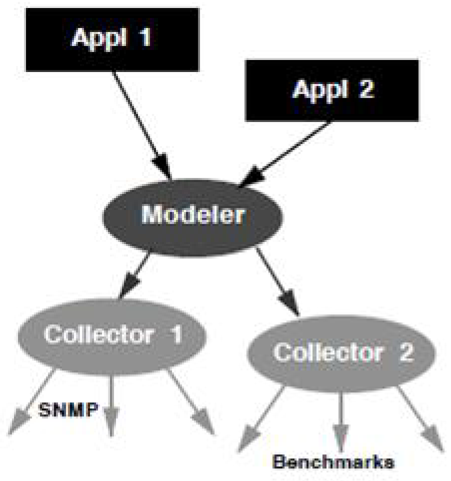


Figure 2.8. REMOS Implemetation

2.2.3.4. Lucent Group

Lucent Group Project aims to model an undirected graph of which nodes are switches and routers and edges are modelled as the direct connection between pairs of interfaces of nodes. In this project, SNMP queries are sent to a router or a switch then IP tables are retrieved. The IP addresses are sent to the topology manager. Topology manager module uses this IP addresses in spanning tree protocol to determine unique forwarding paths for each node. All the paths and IP addresses are stored in a database; periodically mPinger module uses the IP addresses to mping. mPing is a kind of ping operation which is developed by the owners of the study. mPinging is refers to ping from a given source address to a given destination address. It uses the raw socket option and consequently requires root privileges to run. To obtain complete address sets for each interface of each network element, mPings are executed across any pair of network elements in the set of elements in the input administrative domain [13]. The block diagram of the study is shown in Figure 2.9.

2.2.3.5. Study of Lowekamp et al.

Lowekamp et al. improved the REMOS system by adding FDB to the Collector's SNMP module. FDB information, which contains forward tables, is retrieved from routers

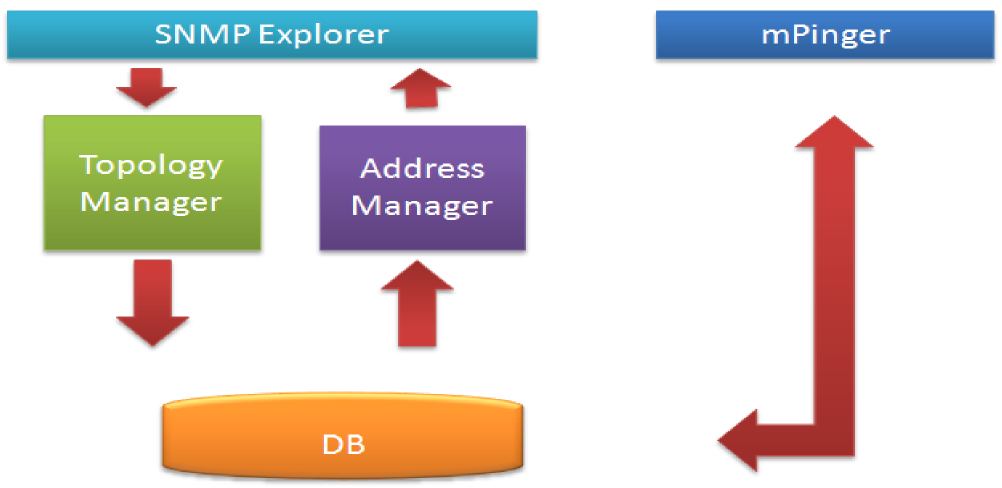


Figure 2.9. Architecture of Lucent Group Project

by using SNMP queries [14].

The topology discovery begins with the set of bridges on the network and downloads the entire FDB from each bridge. The topology derivation is done by traversal from a designated root bridge. After retrieving the FDBs, the root bridge is selected to determine the port of the root to which each bridge is connected. The simple connection theorem is applied to each pair of bridges to determine the port of the root to which it is connected and the child is placed in a set connected to that port. Then the algorithm is called recursively for the set of bridges attached to each port of the root. If a bridge that is directly connected to the previous root bridge is found, and the mapping begins again. The algorithm is shown in Figure 2.10.

After the bridge topology is derived, the endpoints in the network are mapped to their location in the network. Then general endpoint location is determined easily by using Minimum Knowledge Requirement Theorem [14]

Theorem 2.2.1 *(Minimum Knowledge Requirement Theorem) The ports X and Y that connect bridges a and b are uniquely determined if and only if at least one of these conditions is met (Figure 2.11):*

1. *Each bridge has an entry for the other's address in its FDB, not including out of band ports; or*

2. *Bridge a has an entry for b in* F_{a^X} *and* $k \neq X : F_b^Y \bigcap F_{a^k} \neq \emptyset$ *; or*

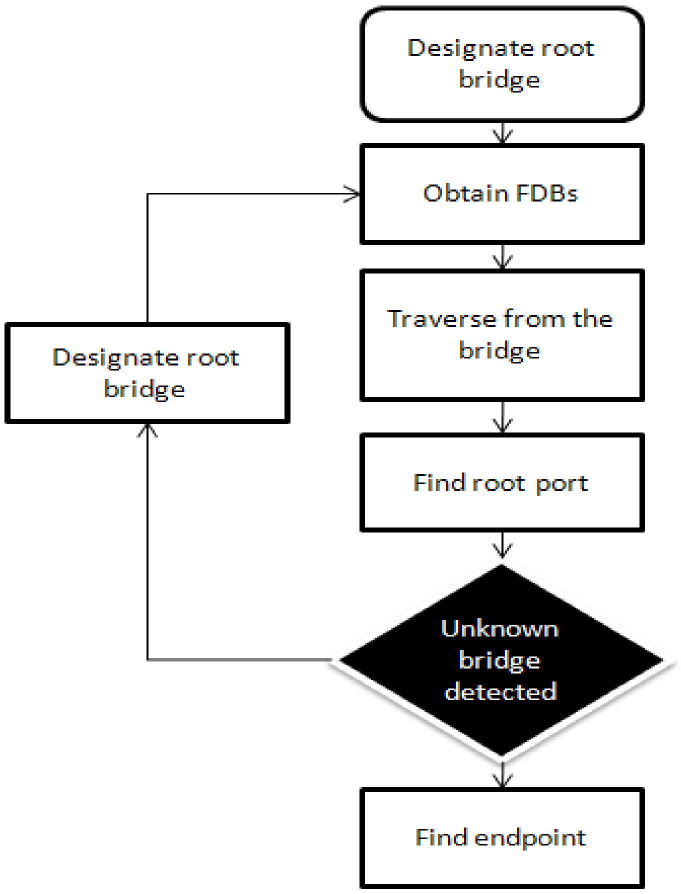


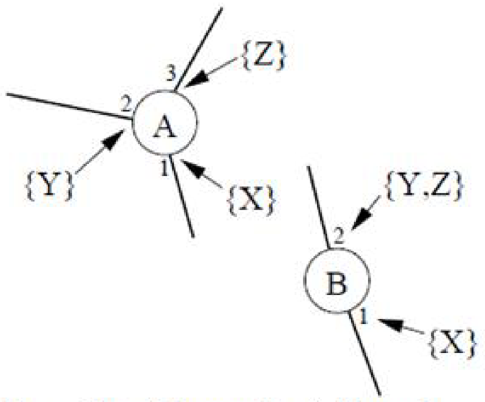
Figure 2.10. Sample

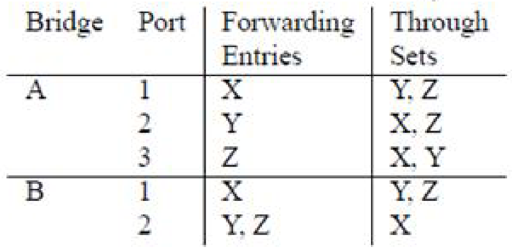
3. *Forwarding entries for three nodes are shared between a and b, divided among at least two ports on one of a or b and three ports on the other bridge. X and Y must be included in those ports. Formally,* i, j , $i \neq j$: $(F_{a^X} \bigcap F_{b^i} \neq \emptyset \bigwedge F_{a^X} \bigcap F_{b^J} \neq \emptyset)$ *and* $k \neq X$: $F_{b^Y} \bigcap F_{a^k} \neq \emptyset$

2.2.3.6. HyNeTD

HyNeTD Project aims to discover the router level topology of IP networks via the IP address spaces and the SNMP community names. HyNeTD finds out the topology in five steps. In the first step, HyNeTD sends a ping towards all the addresses which belong to the discovered addresses spaces.

In the second step HyNeTD looks for available SNMP information sources by





Bridge	Port	Forwarding Entries	Through Sets
A	1	X	Y, Z
	2	Y	X, Z
	3	Z	X, Y
B	1	X	Y, Z
	2	Y, Z	X

Figure 2.11. Sample

sending an SNMP request and waits for a response. If a response arrives the SNMP availability may be assumed. In order to decrease the network load, HyNeTD sends one UDP packet towards the SNMP port of any active addresses. HyNeTD assumes that it is impossible to obtain SNMP information from addresses responding with ICMP “Host Unreachable-Port” messages if the hosts are alive. If an address does not respond, there are three possibilities. First one is that the UDP packet or the ICMP response is lost. The second one is that the device is configured to not respond. The last one is that there is an active SNMP agent.

In the third step, for each address belonging to the SNMP list, HyNeTD sends some SNMP requests to retrieve information belonging to the SNMP Management Information Base (MIB). HyNeTD assumes that all the addresses that do not respond to the first SNMP request cannot be used as SNMP information sources. Therefore, it does not send any future SNMP request toward such addresses, which are also added to the ICMP list. Finally, HyNeTD sends DNS inverse look-up calls to obtain the addresses name.

In the forth step HyNeTD use of both ping-RR and traceroute to obtain information also from routers that do not respond to traceroute, HyNeTD uses a modified traceroute implementation: it uses ICMP Echo-Request messages instead of UDP packets with an invalid destination port to discover some links that traceroute would have not found due to the presence of devices that do not respond to the ICMP Echo-Request.

Finally, during the fifth step, HyNeTD merges and elaborates all retrieved information with the help of Extractor modules [15]. The diagram of the HyNeTD architecture is shown in Figure 2.12 and Figure 2.13.

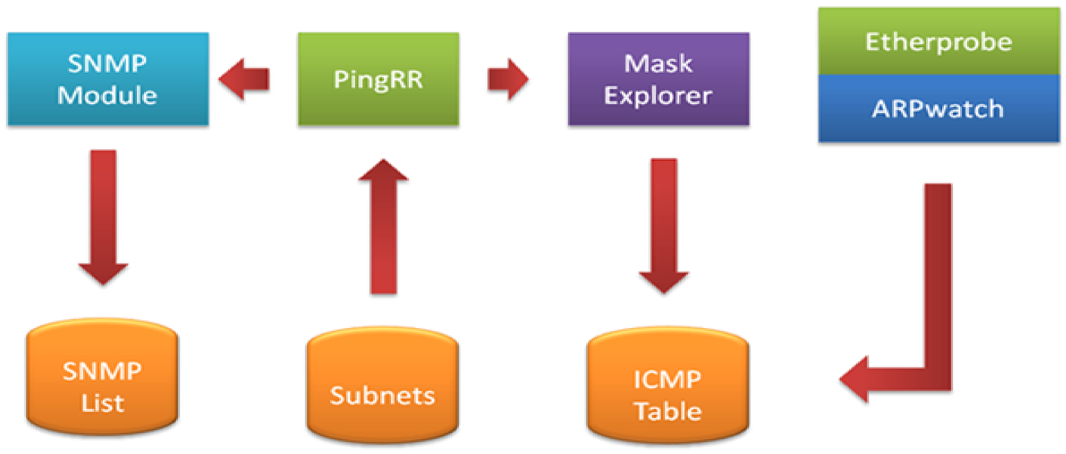


Figure 2.12. Block diagram of HyNeTD Explorer Modules

2.3. Comparisons

To sum up, there are two methods to discover the topology, namely; passive monitoring which listens the packets along with the measurement of benchmarks of the computer network and active probing which sends packets and then listens to the responses. There are many academic studies on discovering computer network topology issue. These studies can be classified into three: benchmark strategy, hop by hop strategy and hybrid strategy [4]. Firstly, the benchmark strategy is the one that uses benchmarks to measure bandwidth across the topology. For instance, Topology-d measures the bandwidth and then builds the minimal spanning tree model of the computer network. The IDMaps uses benchmarks to build distance maps that offer a global view of a computer network such that the distance between any two network hosts can be derived.

Secondly, hop by hop strategy is an approach that uses hop limits. Within the hop by hop strategy, as an instance, The Mercator is a tool to produce an internet map. Internet map is a graph where nodes represent routers on the internet and where links represent adjacencies between routers. Lastly, the hybrid strategy comes as an approach where different methods and strategies can be combined to each other by using different

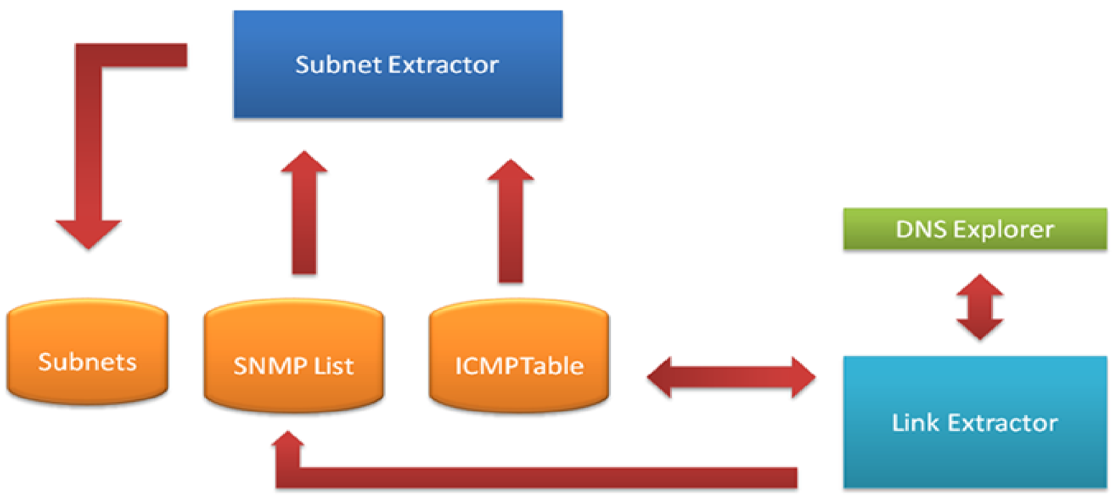


Figure 2.13. Block diagram of HyNeTD Extractor Modules

protocols. As an instance, HyNetD project uses EtherHostProbe, ping and RIPQUERY to discover the computer network topology. As another instance, in REMOS Project, two collectors are used: one is for retrieving data from routers by using SNMP and other one is for monitoring bandwidth to catch the benchmarks. Lowekamp group have studied on upgrading REMOS project; FDB module is added to the project. The comparison table of the studies is shown in Table 2.1.

Table 2.1. Comparison table of the recent studies

Project Name	Limitations
Mercator	Produces internet map
REMOS	Imposes network load
Topology-d	Needs long time to produce map
Theilmann and Rothermel Algorithm	Produces internet map
	Needs long time to produce map

There are two important parameters for the thesis: time and the network components discovered. Time is important because the system must be real time to generate rules for firewall. If the time to discover the topology is long, the time for generating firewall rules will be long as well. The second one is the network components discovered. Most of the studies focus on the routers, switches and the relation between them. Only a few of the studies focus on the discovery of the local subnet(s). However, the network components in local subnets are as important as the others to generate firewall rules. So in order to generate firewall rules a tool which is developed by discovering topology, is needed. In Chapter 3, the topology discovery approach of the thesis will be mentioned.

CHAPTER 3

PROPOSED TOPOLOGY DISCOVERY APPROACH

In this chapter, topology discovery module of the thesis will be explained. There are two sub modules in topology discovery module. In the following subsections, these will be mentioned.

3.1. Discovering IP Addresses

To discover the active IP addresses in the network, there are two important issue: the first one is the structure of the network and the second one is the algorithm used to detect IP addresses. The network which the thesis is designed on, is a campus network. The campus network has one gate for internet including many subnets. The routers and switches connect to both each other and the internet. The structure is shown in Figure 3.1. For the network security, firewalls are located inside the network: some of them are

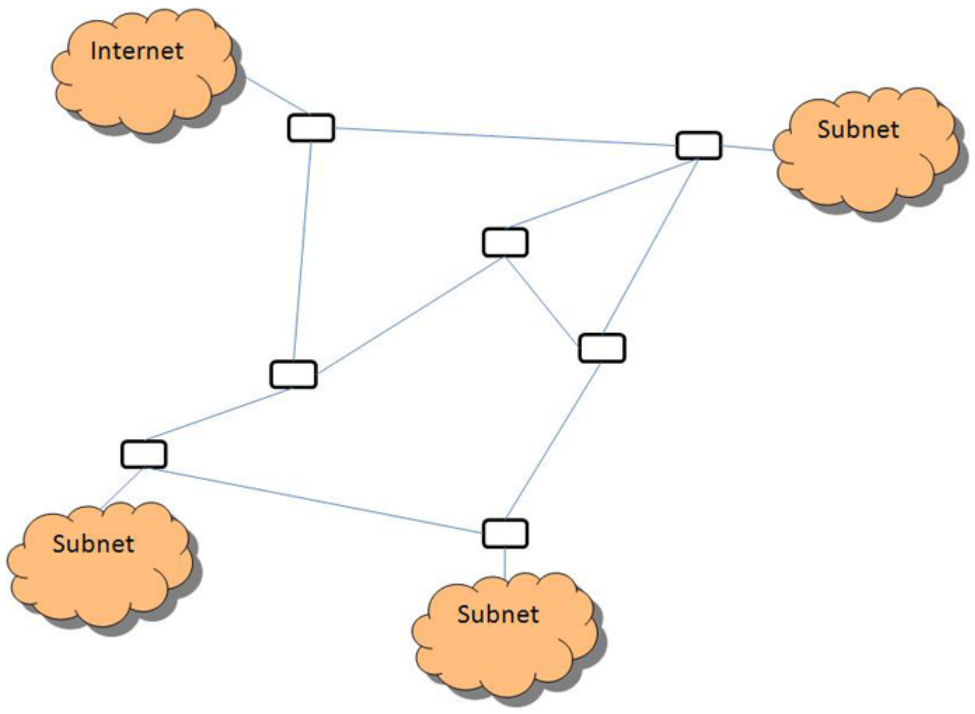


Figure 3.1. Simple Network Example

located at the gates of the subnets and some are located at the place of routers. (Most of the firewalls have ability to route. In the thesis, it is assumed that the firewalls in the network can be used as router.) The structure of network is shown in Figure 3.2. The black nodes in the figure are firewalls and the white ones are any network components such as routers, switches and servers. The second important issue of the IP discovery sub module is

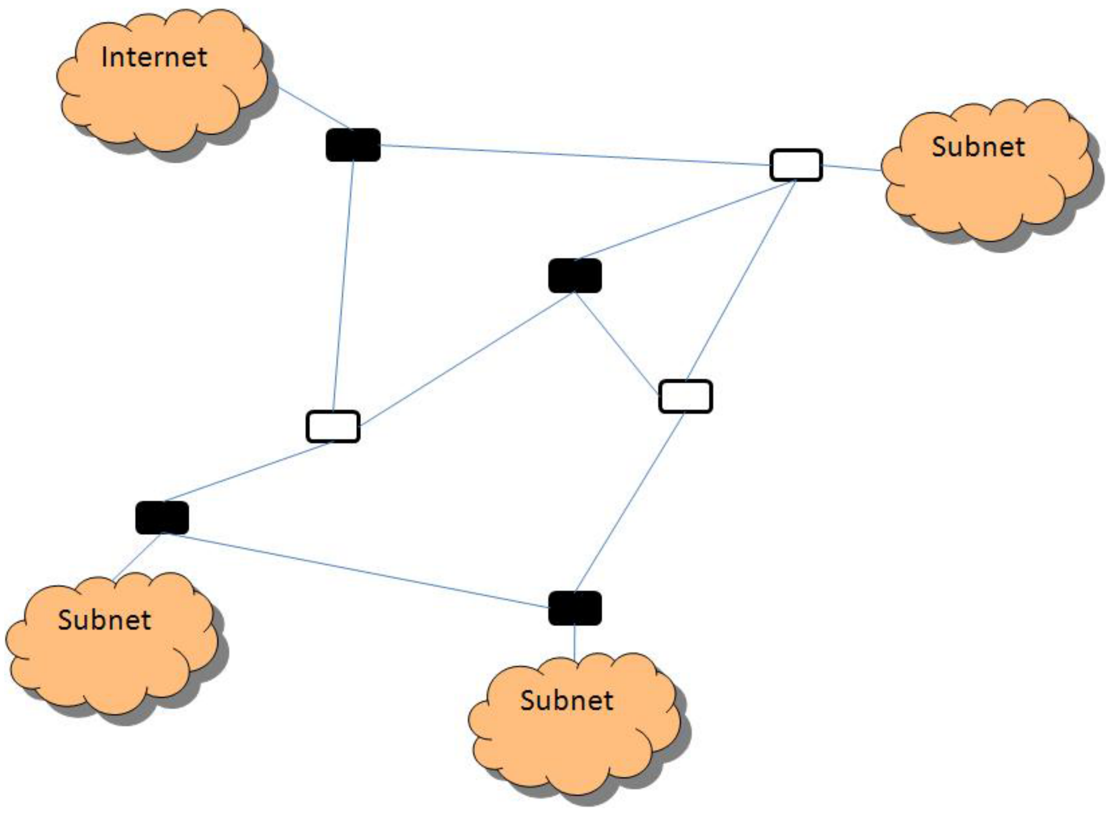


Figure 3.2. Secure Network Example

the algorithm. The algorithm contains three processes. The first process is the traceroute process. In the traceroute process, traceroute packets are sent to the internet addresses which are given as a list periodically to find out the components of the computer network. By doing traceroute, the path of the packet is monitored. In this path, the IP addresses of the network components such as gateways, parent firewalls and routers can be discovered. Sending packets from the same locations to different IP addresses provides different paths, which means more founded network components are discovered. Sending from different locations provides the same effect. The calculation host IP address ranges according to the founded IP addresses is the second process. At first, the network class of the IP address is calculated by using IP address ranges in Table 3.1. For example, the IP address

Table 3.1. IP ranges table of the network classes

Network Class	Min IP Address	Max IP Address	Subnet Mask
A Class	0.0.0.0	127.255.255.255	255.0.0.0
B Class	128.0.0.0	191.255.255.255	255.255.0.0
C Class	192.0.0.0	223.255.255.255	255.255.255.0

10.1.155.195 is in Class A and the default subnet mask is 255.0.0.0. After calculating default subnet mask, the network prefix will be calculated. It is done by a technique in which "AND" operation of the binary versions of IP address and subnet mask are giving the network prefix. For example; there is an IP address 192.168.2.3 and it is in Class C so the default subnet mask is 255.255.255.0. Their binary versions are :

192.168.2.3 = 11000001.10101001.00000010.00000011

255.255.255.0 = 11111111.11111111.11111111.00000000

The calculated network prefix is :

```
              11000001.10101001.00000010.00000011
              11111111.11111111.11111111.00000000
AND ----------------------------------------------------
              11000001.10101001.00000010.00000000
```

11000001.10101001.00000010.00000000 = 192.168.2.0

The third process is pinging to each IP address of the calculated IP address range. This process can be explained by the same example stated above. The network prefix is calculated as 192.168.2.0 and the subnet mask is calculated as 255.255.255.0 by using the subnet masks in Table 3.1. It means that there may be 253 IP addresses because 192.168.2.0 is not used and 192.168.2.255 is for broadcast. The broadcast IP addresses are not used because unless the host which does the ping broadcast is in the same subnet, the broadcast request will not be replied.

In summary, ARP broadcast packets are used to discover the subnet where it is located. Sending and getting reply messages takes less time than doing these by using ping packets. To achieve all operations, it is assumed that the ICMP and ARP protocols are accepted in the firewalls rules.

3.2. Discovering Port Numbers

Port scan is applied to all IP addresses which reply to the ICMP ECHO packet. The port scan refers to sending TCP and UDP packets to all port numbers which range 0 to 65535. Firstly, TCP packets are used to connect to the port. If the connection is unsuccessful, UDP packets are sent to the port. If both of these are unsuccessful for all ports, it means that this IP address belongs to a router or a switch.

To sum up, the thesis algorithms and processes are designed to work with packet filtering firewalls. So there are two important steps to generate a rule. In the first step, traceroute process is used to detect IP address and then calculate the IP ranges. The ping process sends ping packets to all the calculated IP addresses and the ARP process detects the local subnets IP addresses. In the second step, TCP and UDP packets are used to find the open ports.

CHAPTER 4

FIREWALL RULE GENERATION APPROACH

Topology discovery module, which provides alive IP addresses and the open ports of each IP address, is one of the main approaches of the thesis. The firewall rule generation module is the second. It uses alive IP addresses and the open ports. In this module, there are two sub modules: the rule generation sub module and rule transfer sub module. The first sub module generates firewall rules and the second one transfers the generated rules. In the subsections below, these sub modules will be introduced.

4.1. Rule Generation Module Evaluation

Rule generation sub module generates rules by using global and local polices. Policy is the behaviour that can be applied in accordance with a situation. In the subsections following, global and local policies will be mentioned.

4.1.1. Global Policy

Global policy is the policy which is valid in the whole network. For instance, the boss in a company recognizes that using MSN prevents his employees from working efficiently. So, he thus he bans this application. This is applicable for all the people in the company. Because of that, it is included in global policy. When the Network administrator defines this global rule as a global policy, the rules are defined according to the global policy. For example, the port which MSN is used becomes X, then the global policy will be "from any IP to any IP on port X: deny" When a new IP is added to the system, FERGI will rule generate this policy as "from any IP to Y on port X: deny"

The firewall rules have no effect on the rules generated. The only effect is the policies to generate firewall rules. For example, in a workplace where MSN application is banned, some of the employees in the company may reject the rule in that they make use of the MSN in their work relationships such as constructing work meetings with customers. The network administrator also describes the rule on the firewall for the IP addresses in the range as “deny”. In this situation, the IP addresses which are excluded from the range must be accepted. However, if the global policy does not change when a new IP is added to the structure, MSN port on the IP address is blocked by generating rule on firewall.

4.1.2. Local Policy

In general, the local policy is the policy which is defined for the exact subnet. For example, X and Y are the students at a university who are making a study of developing search engine as a doctor’s degree thesis. These students are using a variety of hosts and servers while making their thesis. The connection and the information transfer between these hosts and servers are so intense that the campuses all bandwidth becomes full. In a situation like this, if no rules are generated in firewall, then the hosts and the servers will be blocked. In order to prevent the blocking, the hosts and the servers are going to be collected in a specific IP range or can be added to a specific subnet. The servers and the hosts that X and Y use for the tests of their thesis may change all the time. In this situation, the administrator should be informed consistently and also in accordance with the changing situation firewall should reorganize the rules inside. If there were a tool which administrator could define policies into and the tool generated firewall rules according to the changing structure of the network, the response time and the workload of the administrator would be less.

In detail, the local policies are created to generate more specific rules for a subnet when the global policies and the current firewall rules are inadequate. For example, there is a library subnet in a campus network. Only the staffs who work at the library, connect to internet whereas the students have the access only to the library database through the library network. The rules in the library’s firewall state that the rest of the IP addresses in the range which belong to student computers are permitted to connect to internet. Then an access point is located in the library subnet and the IP addresses of the access point are not in the range of student computers. According to current rules in the firewall, all the computers can be connected to the internet through the access point. If there were a tool of which local policy definition is “from any IP to any IP on port 80: deny”, it would generate a rule which denies the http connection on port 80 from that IP address even if the connection is accepted according to the current rules in the firewall. In shortly, even if the current rules in firewall accepts or denies an operation, the tool decides what to generate according to its local polices.

4.2. Transferring Generated Rules

The rule transferring sub module shares the generated firewall rules and the global policy. The sub module does not store each others' IP addresses. So parent must be searched. The traceroute operation lists the IP addresses of the network components which are located between the destination and the current component. These IP addresses can belong to any routers, gateways or firewalls. For this reason, it is checked whether any IP address in the list belongs to a switch/router or a firewall by trying to connect to its 9099 port. If the IP address belongs to a firewall, the firewall is notified after the rule generation operation for generating new rules according to the transferred rules. For example, if a rule "from any IP to any IP on X: deny" is generated, there must be a rule like "from any IP to Y on X: accept, forward Y" in the parent firewall.

After transferring generated rules, whether the global policy has changed is queried. If the response is positive, global policies are taken by the current component. If the firewall module is busy at the time, the request for global policies is repeated after two minutes. The pseudo code of the scenario is shown in Algorithm 1:

Algorithm 1 Rule Transfer Scenario

```
 1: Set index zero
 2: while index is less than the traceroute IP address list size do
 3:     Set the IP address at index to current IP address
 4:     Connect to current IP address on port 9099
 5:     if connection is failed then
 6:         Increase the index value by one
 7:     else
 8:         Wait for “HELLO” message
 9:         Send generated rules to the IP address
10:         Wait for the “OK” message
11:         Send “HAS GLOBAL POLICY CHANGED” message
12:         Wait for the response
13:         if response message is “YES” then
14:             Send “WAITING” message
15:             Receive global policies
16:             Send “OK” message
17:             Close the connection
18:         else if response message is “NO” then
19:             Send “OK” message
20:             Close the connection
21:         else if response message is ”WAIT“ then
22:             Send ”OK“ message
23:             Sleep for two minutes
24:             Go to step 11
25:         end if
26:         Set the size of traceroute IP list size to index
27:     end if
28: end while
```

CHAPTER 5

IMPLEMENTATION AND TOOL SUPPORT

A tool called FERGI (FirEwall Rule Generation Interface) is developed as the thesis product. In this chapter, the requirements and the implementation steps of the tool will be mentioned.

5.1. Requirements

FERGI has requirements to run. First of all is Java SDK because FERGI is developed in Java. However, in order to produce and capture the packets of Link Layer, which is the second layer of the OSI Model, Jpcap is required. Although Jpcap is platform-independent, it needs different libraries in accordance with the operating system on which it runs such as Winpcap and Libpcap. In the subsections following, these libraries will be introduced in detail.

5.1.1. Jpcap

Jpcap [16] is an open source Java library licensed under GNU LGPL for capturing and sending network packets. It can capture and send Ethernet, IPv4, IPv6, ARP/RARP, TCP, UDP, and ICMPv4 packets through a network in java. Jpcap provides facilities to [16]:

- Capture raw packets live from the wire.
- Save captured packets to an offline file, and read captured packets from an offline file.
- Automatically identify packet types and generate corresponding Java objects (for Ethernet, IPv4, IPv6, ARP/RARP, TCP, UDP, and ICMPv4 packets).
- Filter the packets according to user-specified rules before dispatching them to the application.

- Send raw packets to the network

Jpcap runs on libpcap or winpcap libraries. It is developed in C and Java and is used to develop many kinds of network applications, including [16]:

- network and protocol analysers
- network monitors
- traffic loggers
- traffic generators
- user-level bridges and routers
- network intrusion detection systems (NIDS)
- network scanners
- security tools

Jpcap captures and sends packets independently from the host protocols such as TCP/IP. This means that Jpcap cannot block, filter or manipulate the traffic generated by other programs on the same machine, it simply “sniffs” the packets that transit on the wire. Therefore, it does not provide the appropriate support for applications like traffic shapers, QoS schedulers and personal firewalls [16].

FERGI is developed and tested on a host with Windows 32 bits operating system. To run on Windows platform, Jpcap needs Winpcap library. WinPcap [16] is the industry-standard tool for link-layer network access in Windows environments: it allows applications to capture and transmit network packets bypassing the protocol stack. It also has additional useful features, including kernel-level packet filtering, a network statistics engine and support for remote packet capture [16].

WinPcap consists of a driver that extends the operating system to provide low-level network access, and a library that is used to access the low-level network layers easily. This library also contains the Windows version of the well known libpcap Unix API [16].

5.1.1.1. Setting up Jpcap

In order to setup Jpcap for Windows, the following steps should be followed:

- download jdk 6.0 and install
- download Winpcap and install
- download Jpcap for Windows and install

5.1.1.2. Jpcap Practices

In the subsection, code samples for capturing and sending packets by using Jpcap will be mentioned. To capture and send packets, firstly the network interfaces of the computer are listed. The following sample code obtains the list of network interfaces and prints out their information [16]:

```
//Obtain the list of network interfaces
NetworkInterface[] devices = JpcapCaptor.getDeviceList();

//for each network interface
for (int i = 0; i < devices.length; i++) {
  //print out its name and description
  System.out.println(i+": "+devices[i].name
        + "(" + devices[i].description+")");

  //print out its datalink name and description
  System.out.println(" datalink: "+devices[i].datalink_name
        + "(" + devices[i].datalink_description+")");

  //print out its MAC address
  System.out.print(" MAC address:");
  for (byte b : devices[i].mac_address)
    System.out.print(Integer.toHexString(b&0xff) + ":");
  System.out.println();

  //print out its IP address, subnet mask and broadcast address
  for (NetworkInterfaceAddress a : devices[i].addresses)
    System.out.println(" address:"+a.address
        + " " + a.subnet + " "+ a.broadcast);
}
```

This sample code may show a result like the following (on windows) [16]:

```
0: \Device\NPF_{C3F5996D-FB82-4311-A205-25B7761897B9}
    data link:EN10MB(Ethernet)
    MAC address:0:50:56:c0:0:1:
    address:/fe80:0:0:0:3451:e274:322a:fd9f null null
    address:/172.16.160.1 /255.255.255.0 /255.255.255.255
```

The list of network interfaces is obtained and the type of the network interface is selected to capture packets. Then the selected interface is opened by using JpcapCaptor.openDevice() method. The following piece of code illustrates how to open a network interface.

```
NetworkInterface[] devices = JpcapCaptor.getDeviceList();

//set index of the interface that you want to open.
 index=...;

//Open an interface with openDevice(NetworkInterface intrface,
        int snaplen, boolean promics, int to_ms)
JpcapCaptor captor=
    JpcapCaptor.openDevice(device[index],65535, false, 20);
```

After obtaining an instance of JpcapCaptor, the packets can captured from the interface. There are two major approaches to capture packets using a JpcapCaptor instance. The first one is using call back method. The callback method is implemented as the extender of PacketReciever interface to process captured packets so that Jpcap calls it back every time a packet is captured. In detail, a callback method is implemented by overwriting the receivePacket() method of the PacketReceiver interface. The following code shows a call back method which simply prints out a captured packet [16]:

```
class PacketPrinter implements PacketReceiver {
  //this method is called every time Jpcap captures a packet
  public void receivePacket(Packet packet) {
    //just print out a captured packet
    System.out.println(packet);
  }
}
```

The second approach is capturing packets one by one. According to this approach, the packets are captured using the JpcapCaptor.getPacket() method. The getPacket() method simply returns the captured packet and can be called multiple times to capture consecutive packets. The following code prints out the captured packets by using this approach [16]:

```
JpcapCaptor captor=
    JpcapCaptor.openDevice(device[index],65535, false, 20);

for(int i=0;i<10;i++){
  //capture a single packet and print it out
  System.out.println(captor.getPacket());
}

captor.close();
```

In Jpcap, a filter can be defined so that Jpcap does not capture the unwanted packets. The following code shows a TCP packet filter:

```
JpcapCaptor captor=
    JpcapCaptor.openDevice(device[index],65535, false, 20);
//set a filter to only capture TCP/IPv4 packets
captor.setFilter("ip and tcp", true);
```

FERGI uses Jpcap to send ICMP and ARP packets. The sample codes of FERGI is shown in Appendix A.

5.2. Preliminary Design

In this section, the preliminary design of the FERGI will be introduced. FERGI has two main modules; these are Topology Discovery Module and Rule Generation Module. Both of these modules have access to the network. The topology discovery module sends and captures packets from the network and the second module needs network connection to communicate to other FERGI's and share changes. The first module does not access to network own its own however, it accesses to the network by using Jpcap. Jpcap runs on Winpcap. The block diagram of the structure is shown in Figure 5.1.

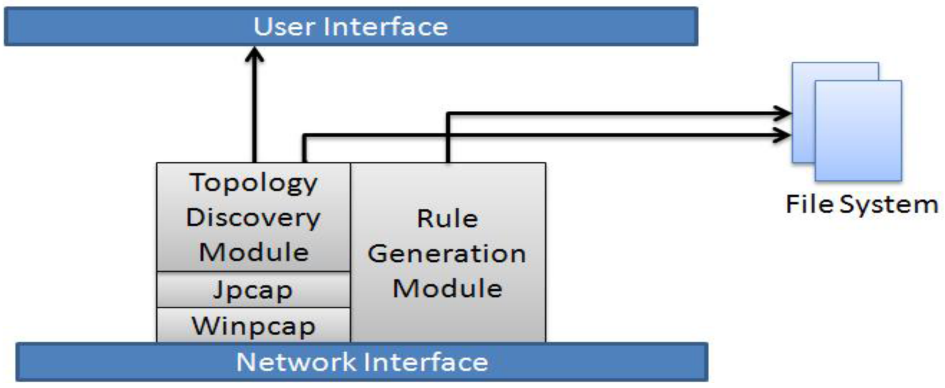


Figure 5.1. Block Diagram of FERGI

5.3. Implementation of the Persistence Layer

FERGI does not use any database to store generated rules or policies in the persistence layer. Because problems may occur cause some problems when there is a need to find out the policies at a time. So FERGI uses four types of files to store data. The first one which is an XML file is for storing the global policies. Here is the sample local policy file:

```
<Policy>
  <action></action>
  <protocol></protocol>
  <type>policyset</type>
  <children>
    <Policy>
      <id>1</id>
      <action></action>
      <protocol></protocol>
      <type>Policy</type>
      <children>
        <Policy>
          <id>rule1</id>
          <srcIP>any</srcIP>
          <srcPort>any</srcPort>
          <dstIP>any</dstIP>
          <dstPort>80</dstPort>
          <action>accept</action>
          <protocol>http</protocol>
          <type>Rule</type>
        </Policy>
      </children>
    </Policy>
  </children>
</Policy>
```

The second file which is for local policies is an XML file, too. The structure of the data in the file is the same as the ones for global policies. However, the global policy differs from the local policy in that the global policy should be only one while the local policy may be a policy set.

The third file is the IP list. FERGI writes the IP addresses which are sent PING packets into a txt file and adds a note about whether the IP address is alive or not next to the IP address in the line. The sample file is shown as:

```
10.1.155.190    false
10.1.155.191    false
```

```
10.1.155.192    false
10.1.155.193    false
10.1.155.194    false
10.1.155.195    true
10.1.155.196    false
```

The fourth file which is an XACML file is for the generated rules. XACML stands for *eXtensible Access Control Markup Language* [17]. It is a declarative access control policy language implemented in XML and a processing model for describing how to interpret the policies. The details of XACML described below [17] as:

- Interchangeable policy format
- Support for fine- and coarse-grained authorization policies
- Conditional authorization
- Policy combination and conflict resolution
- Independence from implementation"

The sample rule file that FERGI generated is shown as:

```
<Rule RuleId="1" Effect="false">
<Description>tcp,any,any,10.1.155.123,80,false</Description>
<Target>
<Subjects>
    <Subject>
        <SubjectMatch
            MatchId="urn:oasis:names:tc:xacml:2.0:
            function:ipAddress-regexp-match">
            <AttributeValue
                DataType="http://www.w3.org/2001/XMLSchema#string">
                ^*:(any)$
            </AttributeValue>
            <SubjectAttributeDesignator
          SubjectCategory="urn:oasis:names:tc:xacml:1.0:
                subject-category:access-subject"
         AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
          DataType="urn:oasis:names:tc:xacml:2.0:data-type:ipAddress"/>
        </SubjectMatch>
    </Subject>
</Subjects>
<Resources>
    <Resource>
        <ResourceMatch
         MatchId="urn:oasis:names:tc:xacml:2.0:function:
            ipAddress-regexp-match">
          <AttributeValue
                DataType="http://www.w3.org/2001/XMLSchema#string">
                ^(10)\.(1)\.(155)\.(123)\:(80)$
          </AttributeValue>
          <ResourceAttributeDesignator
          AttributeId="urn:oasis:names:tc:xacml:
            1.0:resource:resource-id"
          DataType="urn:oasis:names:tc:xacml:
            2.0:data-type:ipAddress"/>
        </ResourceMatch>
    </Resource>
</Resources>
</Target>
<Condition>
    <Apply
    FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
    <Apply
    FunctionId="urn:oasis:names:tc:xacml:
        1.0:function:string-one-and-only">
    <SubjectAttributeDesignator AttributeId="protocol"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
<Apply>
    <AttributeValue
    DataType="http://www.w3.org/2001/XMLSchema#string">
        tcp
    </AttributeValue>
</Apply>
</Condition>
</Rule>
</Policy>
</PolicySet>
```

5.4. Implementation of the Business Logic

FERGI application is designed on a Kernel object. The Kernel object is similar to the kernel in an operating system. The Kernel object starts and controls all the processes. There are two main modules which cover all the processes.

The first one is the topology discovery module. This module has a manager class called JpcapNetDiscoveryManager which has two important tasks. The first one is to organize the packet processor sub module. The Kernel object calls JpcapNetDiscoveryManager object then the manager object makes the packet generators started. JpcapNetDiscoveryManager object creates the NetworkDevice object according to selected network card then creates the scheduler thread pool with the size of given “max_thread” value which is defined in the settings file.

There are three scheduler threads: traceroute scheduler, ARP scheduler and ICMP scheduler. JpcapNetDiscoveryManager object starts the traceroute scheduler firstly. When the traceroute operation ends, the other two schedulers are started to run at the same time. IP addresses in the traceroute IP list are sent to the ICMPScheduler one by one. ICMPScheduler is the class which calculates the default subnet mask and the IP range accordingly sending ICMP ECHO packets to each IP address in the range. While ICMP-Scheduler object is running, ARPScheduler sends ARP Broadcast packets. The pseudo code of the ICMPScheduler scenario is shown in Algorithm 2:

Algorithm 2 ICMPScheduler Scenario

```
 1: Get the traceroute IP address list
 2: while There are IP addresses in the list do
 3:     Calculate network class of the IP address
 4:     Calculate default network mask
 5:     Calculate the IP range
 6:     while There are IP address in the range do
 7:         Send ICMP ECHO packet to the IP address
 8:     end while
 9: end while
10: Sleep for the period
```

The second important task of the JpcapNetDiscoveryManager is to start the packet sniffer sub module. JpcapNetDiscoveryManager object starts PacketListener object. PacketListener is the packet capturer class which controls sniffs the selected network card. If any packet is received, PacketListener forwards it to PacketProcessor object. PacketProcessor filters the packets whether they are ICMP and ARP or not. The pseudo code of the packet sniffer sub module is shown in Algorithm 3:

The second main module of FERGI is the rule generation module. In this module, the ServiceListener object organizes the rule generation and takes the alive IP addresses then does the port scan. It also calls the XACMLRuleGenerator object to generate rule according to founded alive IP and open port numbers. The scenario is shown in Figure 5.2.

The class diagram of the whole scenario is shown in Figure 5.3. The Kernel object also calls the NetworkListener object which is the interface providing processes to communicate with each other. NetworkListener has observers. When any message or situation occurs, it passes the event to its observers. The observers of NetworkListener

Algorithm 3 Packet Sniffing Scenario

```
1:  Start PacketListener
2:  while PacketListener is active do
3:    if A packet is recieved then
4:      Forward packet to PacketProcessor
5:      if the packet is an ARP packet then
6:        Create Host object
7:        Forward it to ServiceListener
8:      else if the packet is an ICMP packet then
9:        Create Host object
10:       Forward it to ServiceListener
11:     end if
12:   end if
13: end while
```

are the user interface object and ServiceListener object. The class diagram of the classes is shown in Figure 5.4.

5.5. Implementation of the Presentation Layer

FERGI has a home page and all the operations and configurations are done on this frame. The main page of FERGI is shown in Figure 5.5. The main page has three parts: tool-bar menu, task-bar menu and monitor panel.

The first part of the frame is the tool-bar menu. All the buttons for operations such as start and stop scanning are on the tool-bar. The “Start Scan” button notifies the FERGI Kernel to start the processes of both the topology discovery and the rule generation. After the discovery generation module is started, “Start Scan” button becomes inactive and “Stop Scan” button becomes active. This is shown in Figure 5.6.

Another button on the tool bar menu is the “Log viewer” button. FERGI logs each operation such as the statuses of operations, results of operations and IP addresses tested. The button shows these log information in another frame. A screen shot of “Log viewer” is shown in Figure 5.8.

The last button on the tool-bar menu is the “About Fergi” button which shows general information about FERGI. The screen shot of it is shown in Figure 5.8.

The second part of the main frame is the task-bar menu. The buttons in the task-bar are for users to do configurations of the system. The “Network Interface Configurations” button which is the first part of the task-bar is for network card selection. There are two selections: the first one is for inner network and the second one is for outer network. These two network cards can be either the same network card or not. The details of the

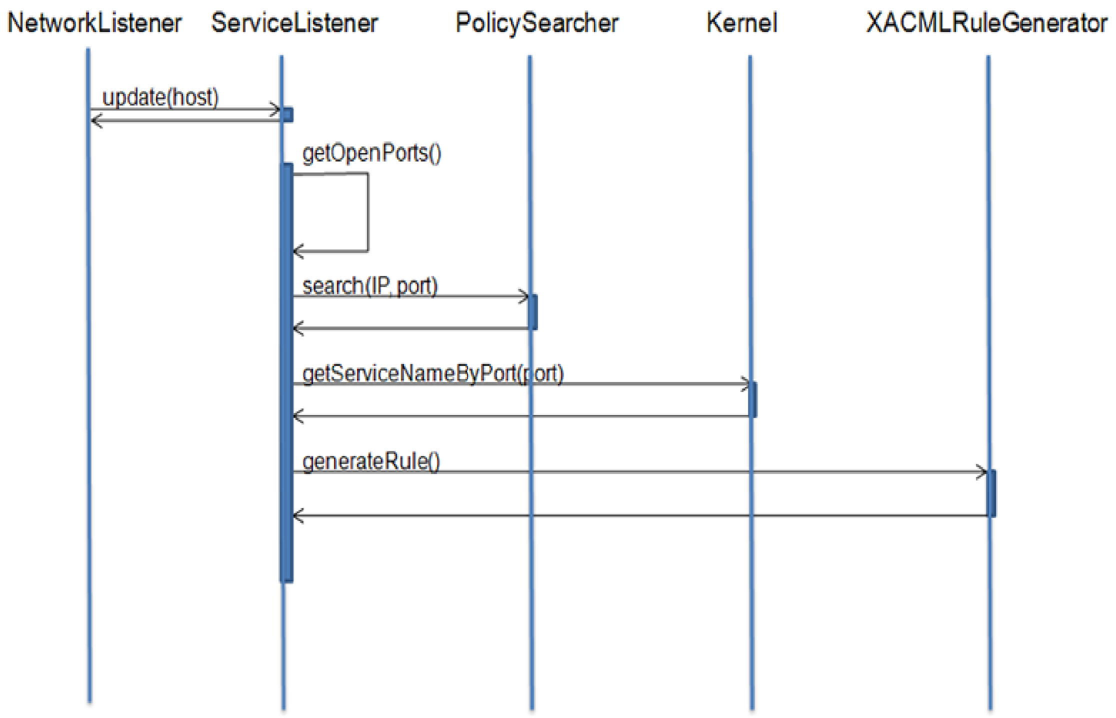


Figure 5.2. Scenario of Rule Generation

network cards on the machine which FERGI runs on are listed and then one of these is chosen by the user. The screen shot of "Network Interface Configurations" is shown in Figure 5.9.

The network interface configuration is necessary to start topology discovery operation. Before selecting the network card, packets cannot be sent and captured. If no network card has been selected, a warning message appears on the main frame. The screen shot of the waning message is as shown in Figure 5.10.

The second part of the task-bar is the "Settings" button. When it is clicked, a new frame is opened. There are three fields on the frame: the listen port field, the period field and the scheduler size field. The "Listen Port" field which is the first field on the "Settings" frame is for the port number which FERGI uses to communicate to another FERGI. The second field on the "Settings" frame is the "Period" field. It is for the value of the topology discovery module period. Topology discovery module runs periodically according to the value. The last field on the "Settings" frame is "Scheduler Size" field. The scheduler size is the size of scheduler pool. When a scheduler process is created, it is forwarded to the scheduler pool to run. System cannot make more processes run. The screen shot of the "Settings" frame is shown in Figure 5.11.

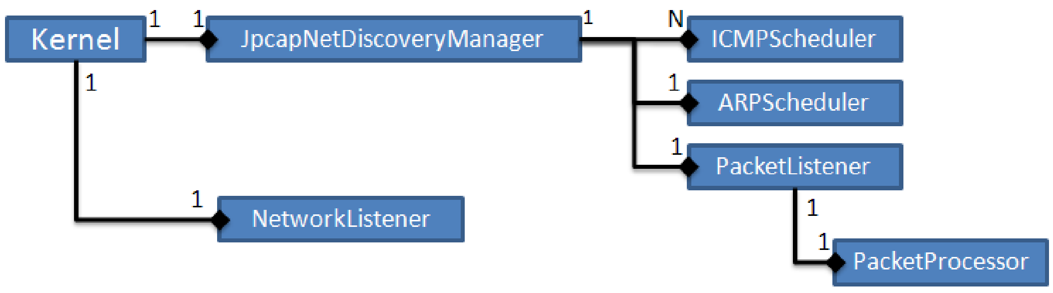


Figure 5.3. Class Diagram of Relation between Kernel and Processes

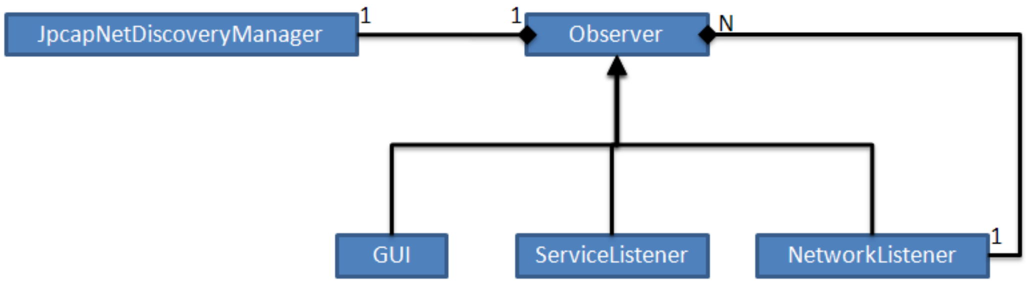


Figure 5.4. Class Diagram which shows how processes communicate to each other

The last two buttons on the task-bar are “Global Policy Configuration” button and “Local Policy Configuration” button. They use the same frame which is shown in Figure 5.12. As shown in figure, policies are the set of rules. Rules and policies can be reorganized by using the three buttons which are located on the south of the frame.

The last part of the main frame is the “Monitor Panel” on which the active IP addresses are seen. The screen shot of the monitor panel is shown in Figure 5.13.

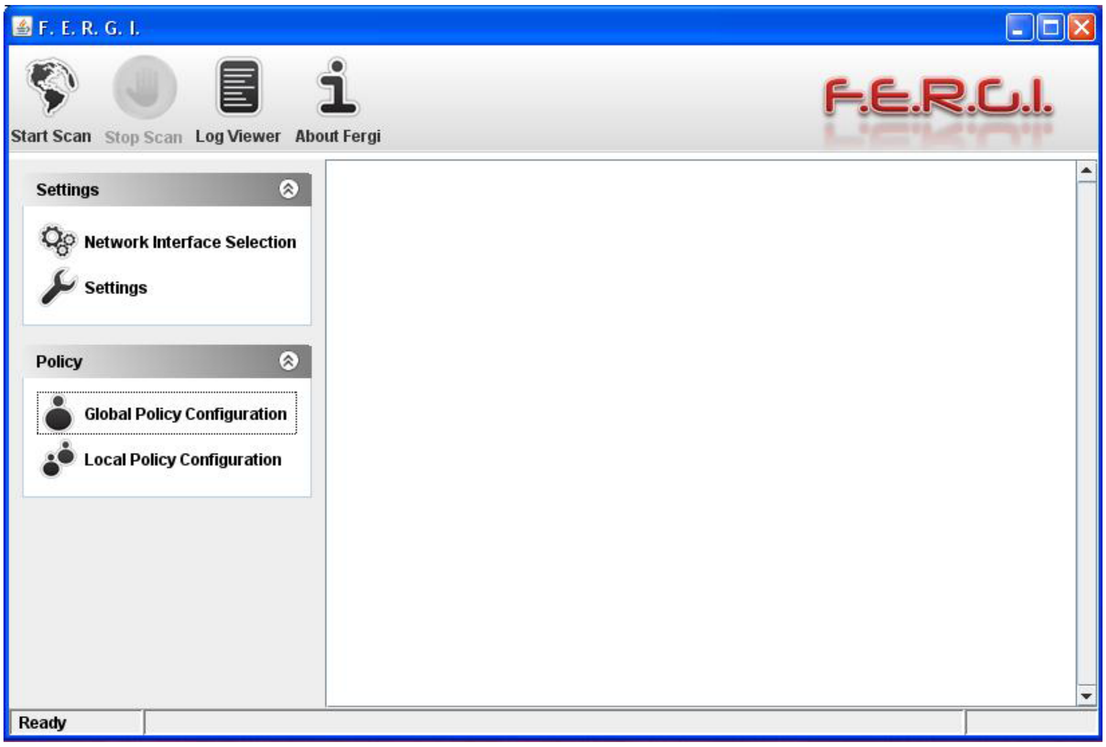
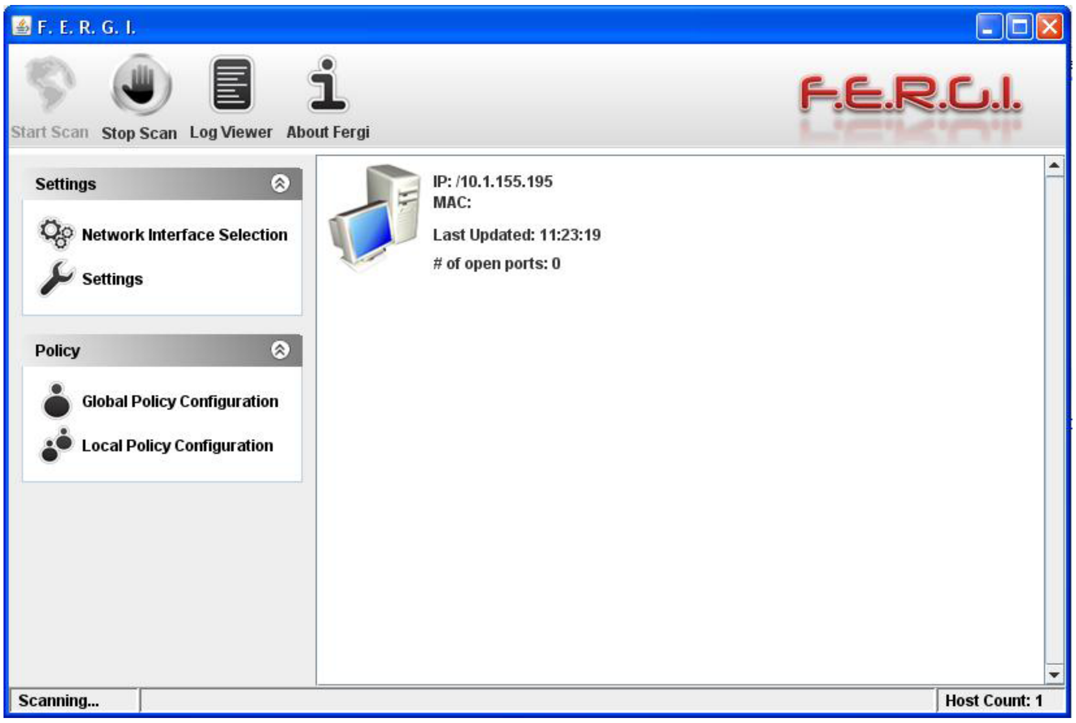


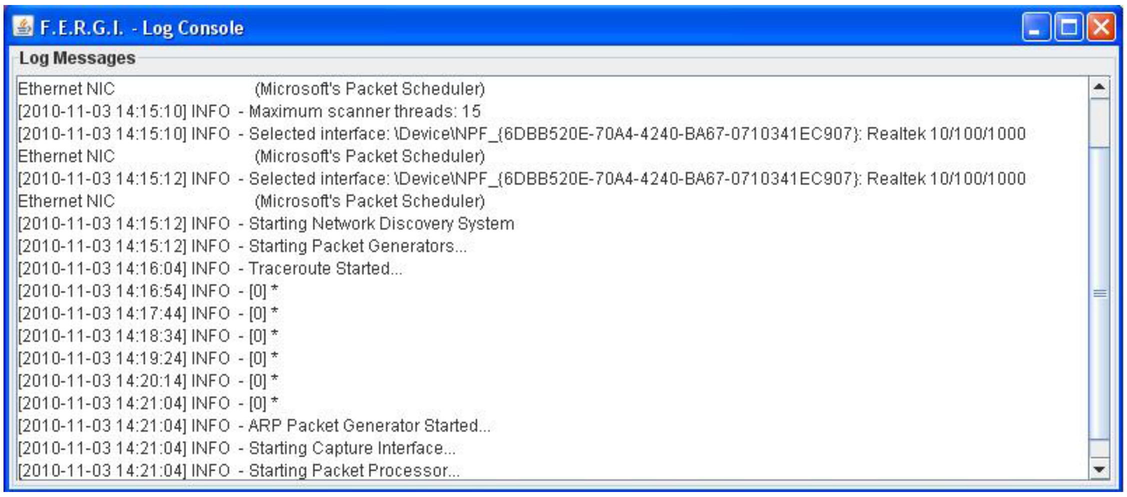
Figure 5.5. FERGI Main Frame



IP: /10.1.155.195
MAC:
Last Updated: 11:23:19
# of open ports: 0

Scanning...
Host Count: 1

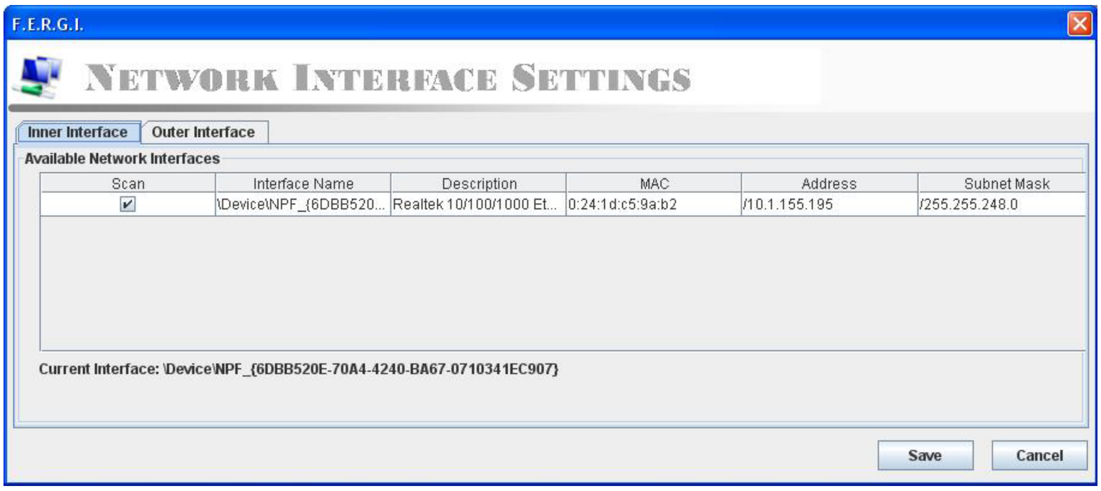
Figure 5.6. Screen shot when FERGI is scanning



```
Ethernet NIC                    (Microsoft's Packet Scheduler)
[2010-11-03 14:15:10] INFO  - Maximum scanner threads: 15
[2010-11-03 14:15:10] INFO  - Selected interface: \Device\NPF_{6DBB520E-70A4-4240-BA67-0710341EC907}: Realtek 10/100/1000
Ethernet NIC                    (Microsoft's Packet Scheduler)
[2010-11-03 14:15:12] INFO  - Selected interface: \Device\NPF_{6DBB520E-70A4-4240-BA67-0710341EC907}: Realtek 10/100/1000
Ethernet NIC                    (Microsoft's Packet Scheduler)
[2010-11-03 14:15:12] INFO  - Starting Network Discovery System
[2010-11-03 14:15:12] INFO  - Starting Packet Generators...
[2010-11-03 14:16:04] INFO  - Traceroute Started...
[2010-11-03 14:16:54] INFO  - [0] *
[2010-11-03 14:17:44] INFO  - [0] *
[2010-11-03 14:18:34] INFO  - [0] *
[2010-11-03 14:19:24] INFO  - [0] *
[2010-11-03 14:20:14] INFO  - [0] *
[2010-11-03 14:21:04] INFO  - [0] *
[2010-11-03 14:21:04] INFO  - ARP Packet Generator Started...
[2010-11-03 14:21:04] INFO  - Starting Capture Interface...
[2010-11-03 14:21:04] INFO  - Starting Packet Processor...
```

Figure 5.7. Screen shot of Log Viewer

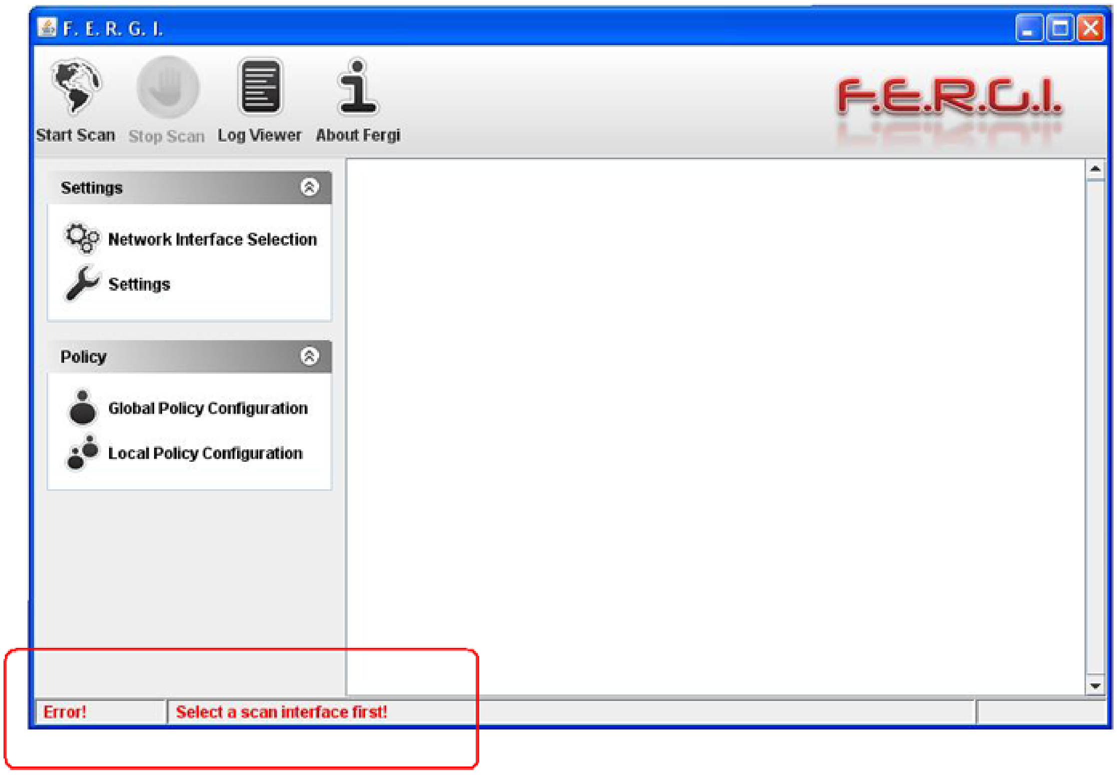
Figure 5.8. Screen shot of 'About FERGI' Frame



Scan	Interface Name	Description	MAC	Address	Subnet Mask
✓	\Device\NPF_{6DBB520...	Realtek 10/100/1000 Et...	0:24:1d:c5:9a:b2	/10.1.155.195	/255.255.248.0

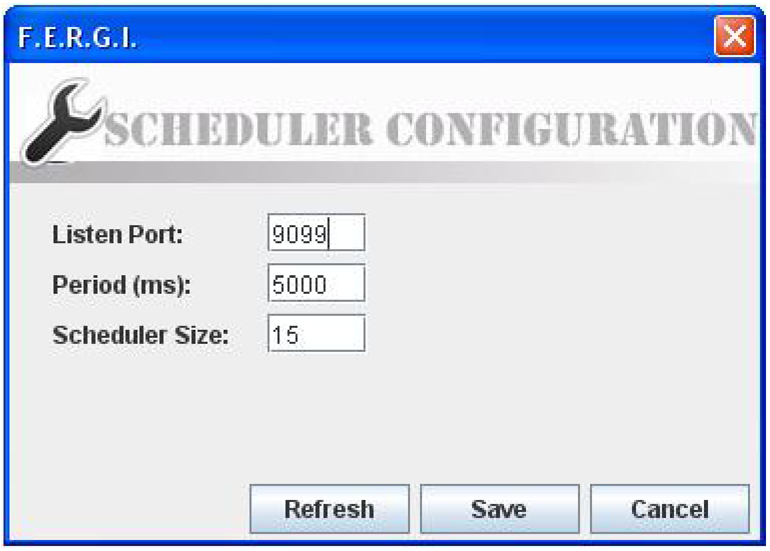
Current Interface: \Device\NPF_{6DBB520E-70A4-4240-BA67-0710341EC907}

Figure 5.9. Screen shot of Network Interface Configuration



Error! Select a scan interface first!

Figure 5.10. Screen shot of Warning Message

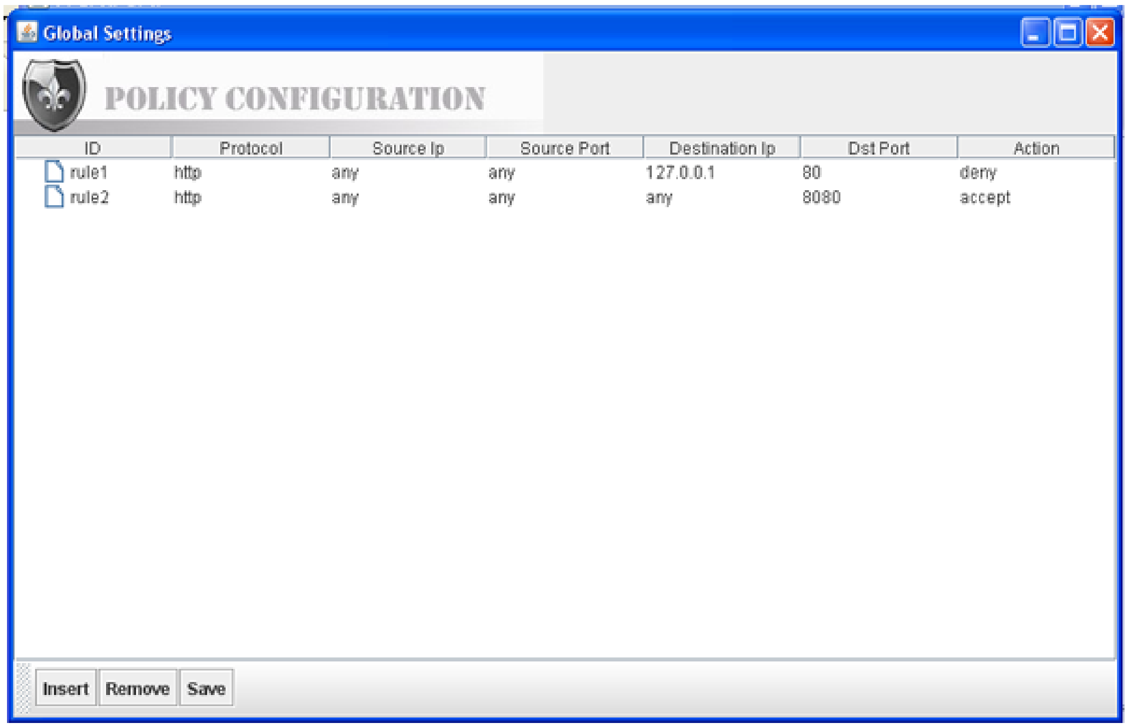


Listen Port: 9099

Period (ms): 5000

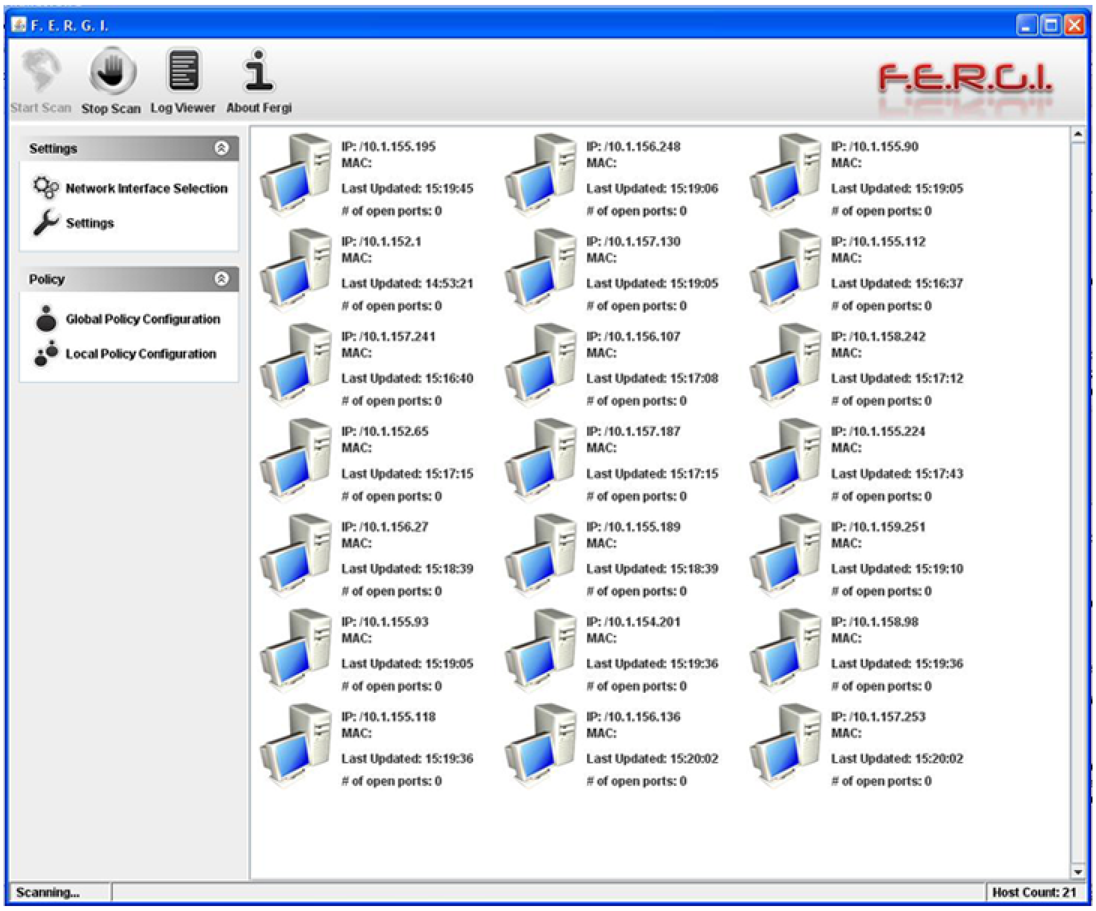
Scheduler Size: 15

Figure 5.11. Screen shot of Settings



ID	Protocol	Source Ip	Source Port	Destination Ip	Dst Port	Action
rule1	http	any	any	127.0.0.1	80	deny
rule2	http	any	any	any	8080	accept

Figure 5.12. Screen shot of Policy Configuration



```
IP: /10.1.155.195            IP: /10.1.156.248            IP: /10.1.155.90
MAC:                         MAC:                         MAC:
Last Updated: 15:19:45       Last Updated: 15:19:06       Last Updated: 15:19:05
# of open ports: 0           # of open ports: 0           # of open ports: 0

IP: /10.1.152.1              IP: /10.1.157.130            IP: /10.1.155.112
MAC:                         MAC:                         MAC:
Last Updated: 14:53:21       Last Updated: 15:19:05       Last Updated: 15:16:37
# of open ports: 0           # of open ports: 0           # of open ports: 0

IP: /10.1.157.241            IP: /10.1.156.107            IP: /10.1.158.242
MAC:                         MAC:                         MAC:
Last Updated: 15:16:40       Last Updated: 15:17:08       Last Updated: 15:17:12
# of open ports: 0           # of open ports: 0           # of open ports: 0

IP: /10.1.152.65             IP: /10.1.157.187            IP: /10.1.155.224
MAC:                         MAC:                         MAC:
Last Updated: 15:17:15       Last Updated: 15:17:15       Last Updated: 15:17:43
# of open ports: 0           # of open ports: 0           # of open ports: 0

IP: /10.1.156.27             IP: /10.1.155.189            IP: /10.1.159.251
MAC:                         MAC:                         MAC:
Last Updated: 15:18:39       Last Updated: 15:18:39       Last Updated: 15:19:10
# of open ports: 0           # of open ports: 0           # of open ports: 0

IP: /10.1.155.93             IP: /10.1.154.201            IP: /10.1.158.98
MAC:                         MAC:                         MAC:
Last Updated: 15:19:05       Last Updated: 15:19:36       Last Updated: 15:19:36
# of open ports: 0           # of open ports: 0           # of open ports: 0

IP: /10.1.155.118            IP: /10.1.156.136            IP: /10.1.157.253
MAC:                         MAC:                         MAC:
Last Updated: 15:19:36       Last Updated: 15:20:02       Last Updated: 15:20:02
# of open ports: 0           # of open ports: 0           # of open ports: 0

Scanning...                                               Host Count: 21
```

Figure 5.13. Screen shot of Monitor Panel

CHAPTER 6

CASE STUDY

In this chapter, the tests for finding out the parameters that may effect FERGI and the validity of FERGI will be mentioned. FERGI is a tool which generates rules for firewalls according to the changing structure of a campus computer network. FERGI monitors the computer network periodically to catch the changes in the network and if it finds out any, it generates a rule according to given policies. All the operations from the beginning of topology discovery until the end of rule generation, are affected by many parameters such as number of network components, components' status that shows whether components being busy or not. The parameters do not need to be only about the network status, they can be about FERGI's own configuration such as period, thread pool size which are for scheduler threads. In this chapter, FERGI's analysis and performance results will be discussed with parameters which affect them by separating FERGI modules and sub modules.

FERGI contains two main modules: topology discovery module and rule generation module. Topology discovery module has two sub modules. These are ARP scheduler and ICMP scheduler modules. ARP scheduler module sends ARP broadcast packets periodically. It is tested whether any change in period of ARP scheduler has an effect on the number of IP addresses discovered. The result is that there is no relation between period and number of components (Figure 6.1).

The status of a network component which is another configuration parameter. Status of a network component shows whether the component is busy or available. A subnet which has four hosts has been tested during fifteen hours. To make the hosts in the subnet busy, files such as movies and music albums were downloaded by using a peer to peer file sharing tool. However, the result of the test showed that there was no relation between the status of hosts and the number of active hosts discovered (Figure 6.2).

While the ARP scheduler period and the status of the network components affect the ARP scheduler, the thread pool size does not have effect on it . The reason why the thread pool size does not affect ARP scheduler is that only one ARP scheduler is created at the beginning of all the processes.

There is a significant relationship between the number of the active hosts in a computer network and the time. While there are sixty six active hosts in the network

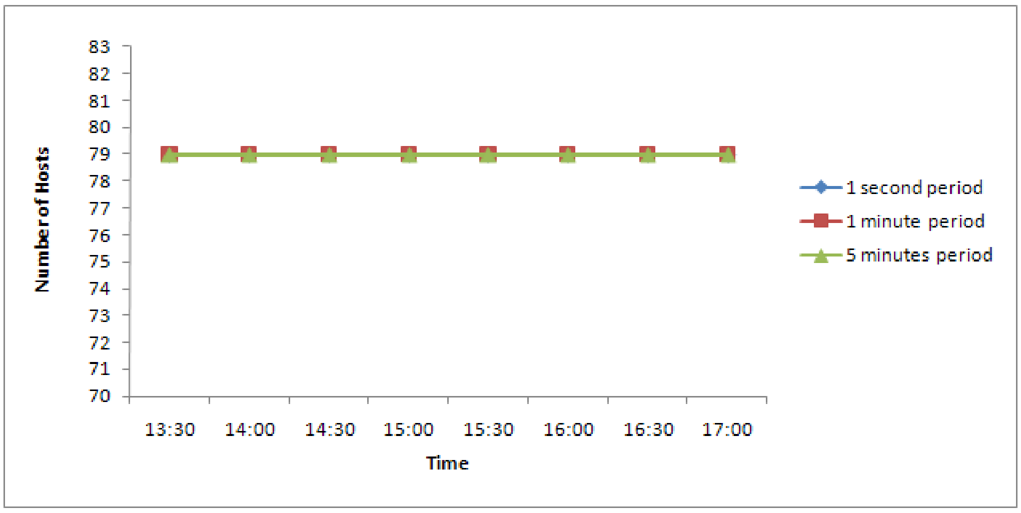


Figure 6.1. The Relation between ARP Scheduler Period and Number of IP Addresses Found

during work hours (Figure 6.1) , there are only one active hosts in the out of the work hours (Figure 6.2).

The second sub module of the topology discovery is ICMP scheduler. The ICMP scheduler cannot be affected by period and status of the hosts because any server or host can reply ping packets even if it is busy. The effect of the thread pool size on the ICMP scheduler is tested. FERGI detects 79 active hosts in a selected network in an hour. When the thread pool size is changed, the number of active hosts discovered by FERGI is monitored. The number of the active hosts decreases when thread pool size is thirty or less (Figure 6.3). The reason of this decrease is that the ICMP schedulers wait for the thread pool to become empty when the pool is full.

The validity of FERGI is tested. In order to find out the number f the active hosts FERGI discovers, FERGI is monitored for an hour during each work day. It was found that the number of the active hosts FERGI detects was 79 for each day. On the other hand, the number of the active hosts was 658 (Figure 6.4). The reason of this finding is that FERGI may not detect all the network components such as routers, switches and the network components which do not reply ICMP ECHO packets. In addition to it, because of the reason FERGI cannot detect switches and routers, the components in the subnets of the switches ad the routers cannot be detected.

While the topology discovery module forwards IP and port numbers discovered to the rule generation module, the rule generation module searches for any policy for the given data in the global and the local policies. The rule generation module is tested by the

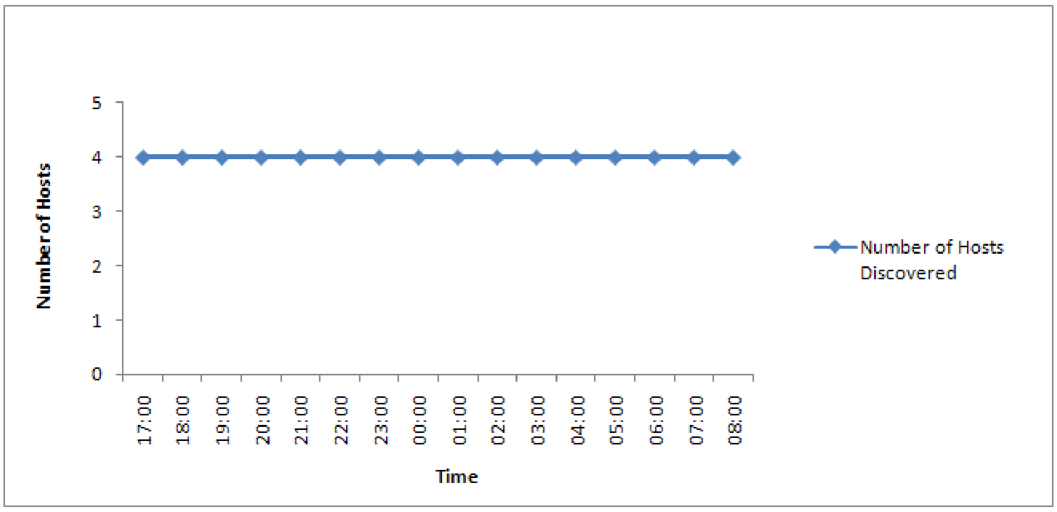


Figure 6.2. The Relation between Number of Hosts Discovered and The Status of Hosts

global and local policies (Appendix B). And the rules generated by the module is shown in Appendix B. The human readable forms of the files are shown below :

- Global Policy
  - tcp,any,any,any,80,deny
  - tcp,any,any,any,8080,accept
- Local Policy
  - tcp,any,any,any,80,accept
- Generated Rules
  - tcp,any,any,10.1.155.80,80,deny
  - tcp,any,any,10.1.158.90,8080,accept
  - tcp,any,any,10.1.154.217,80,deny
  - tcp,any,any,10.1.156.217,8080,accept
  - tcp,any,any,192.168.2.3,80,accept

To sum up, FERGI has an exponential relationship with the number of the components since the network components such as switches and new generation routers do not reply the ICMP ECHO packets. As FERGI is affected by parameters about the computer

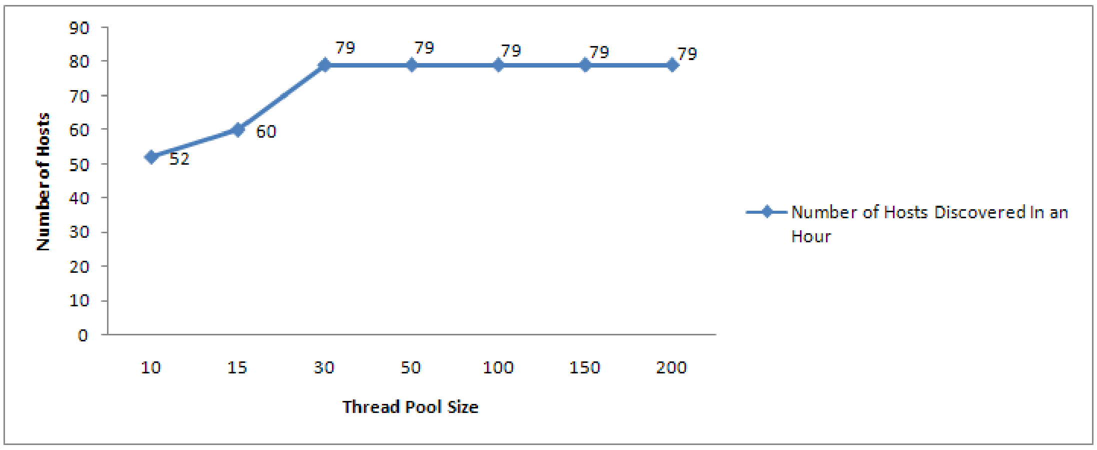


Figure 6.3. The Relation between The Thread Pool Size and The Number of Hosts Discovered

network, its own configuration parameters also affect FERGI. There is an exponential relationship between the thread pool size and FERGI's performance. The period and the status of the hosts do not affect FERGI. In the conclusion chapter, the results will be considered and some inferences will be discussed accordingly.

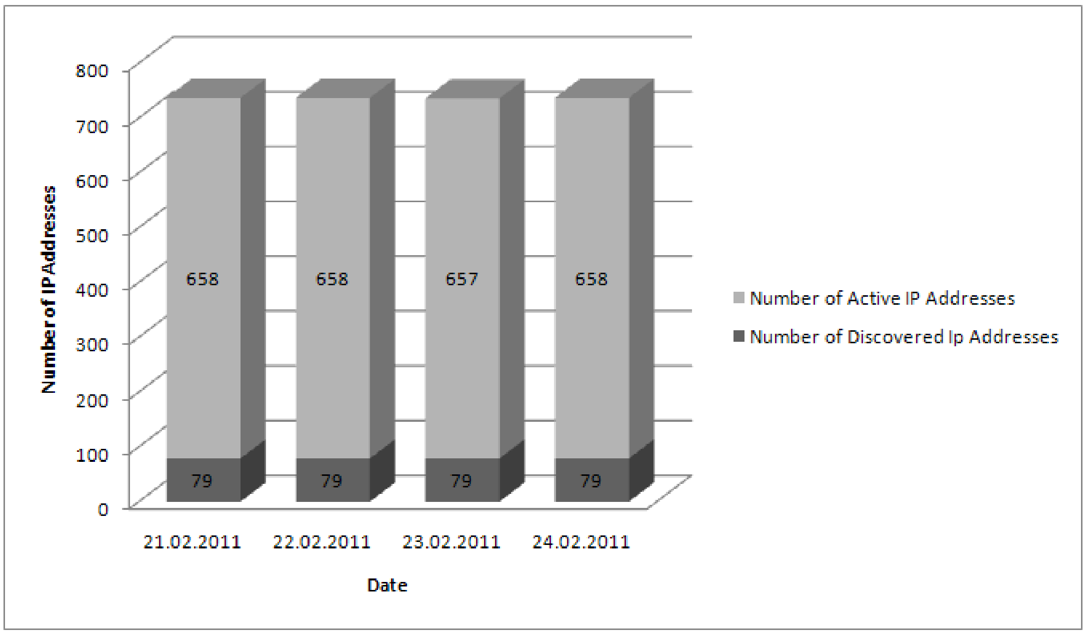


Figure 6.4. The Relation between The Active Network Components in The Campus Network and The Discovered Network Components

CHAPTER 7

CONCLUSION

The discovery of the components of a large computer network (i.e. a campus computer network) along with the generation of rules for firewalls within the network put forward as the aim of this thesis to further improve the network security. Thus, a tool named FERGI (FirEwall Rule Generation Interface) is developed where it discovers the subnet in which it is located by sending ARP Broadcast packets. FERGI discovers parent subnets by sending ICMP ECHO packets to each IP address in the possible IP range which is calculated according to the hop nodes that are found out by traceroute. FERGI does port scan to each IP address to discover the open ports. FERGI uses the data of IP address and open ports to generate firewall rules according to the global policy and local policy. Global Policy is the policy which is valid for whole computer network and local policy is the one that contains rules specific for the subnet. Furthermore, FERGI shares the rules generated to its parents. FERGI repeats all the operations it does periodically to detect any change in the computer network.

FERGI can discover some of the IP addresses which are located between itself and the root gateway at the Internet outgoing point. FERGI can also discover the IP addresses in its own subnet. As an essential point, it is assumed that in the whole computer network, ICMP ECHO and ARP packets are not blocked. FERGI cannot discover all the IP addresses since the network components such as switches and hubs do not reply. ICMP ECHO packets and FERGI cannot discover neighbours' subnets since it does not know the inner IP addresses of its neighbours. Thus, it is suggested that most of the FERGI modules should be located in leaf subnets to properly monitor the whole network; however there should be FERGI modules in the parent subnets, as well.

FERGI is experimented in a small university. It found out all subnets and suggested changes for their firewall policies. Obtained results show that if network monitoring part of FERGI is enhanced with SNMP, all host in the campus network can be included into the security policies.

As a future work, an SNMP (Simple Network Management Protocol) module can be added to the FERGI's system. By using SNMP queries, complete routing tables that contains next hops can be obtained. So the module may help FERGI to discover the switches and also IP addresses in the subnet of the switches. The addition of SNMP module may decrease the network load because of the traceroute and ICMP ECHO packets. The t-spanner algorithm should be used to detect the direct links between network components.

REFERENCES

[1] M. Stemm, R. Katz, and S. Seshan. A network measurement architecture for adaptive applications. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 285–294. IEEE, 2002.

[2] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. On the placement of internet instrumentation. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 295–304. IEEE, 2002.

[3] M. Brodie, I. Rish, S. Ma, G. Grabarnik, and N. Odintsova. Active probing. 2002.

[4] B. Lowekamp, N. Miller, T. Gross, P. Steenkiste, J. Subhlok, and D. Sutherland. A resource query interface for network-aware applications. *Cluster Computing*, 2(2):139–151, 1999.

[5] K. Obraczka and G. Gheorghiu. The performance of a service for network-aware applications. In *Proceedings of the SIGMETRICS symposium on Parallel and distributed tools*, page 91. ACM, 1998.

[6] Uppsala Universitet. Waxman random network topology generator. (http://www.math.uu.se/research/telecom/software/stgraphs.html), October 2010.

[7] M.B. Doar. A better model for generating test networks. In *Global Telecommunications Conference, 1996. GLOBECOM'96.'Communications: The Key to Global Prosperity*, pages 86–93. IEEE, 2002.

[8] W. Theilmann and K. Rothermel. Dynamic distance maps of the internet. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 275–284. IEEE, 2002.

[9] R. Govindan and H. Tangmunarunkit. Heuristics for Internet map discovery. In *IN-*

FOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, volume 3, pages 1371–1380. IEEE, 2002.

[10] M.S. David, D.H. Goldstein, R.K. Neves, and D.C.M. Wood. An Architecture for Discovering and Visualizing Characteristics of Large Internets. 1991.

[11] D.C.M. Wood, S.S. Coleman, and M.F. Schwartz. Fremont: A system for discovering network characteristics and problems. In *Proceedings of the USENIX Winter Conference*, pages 335–348. Citeseer, 1993.

[12] E. KARAARSLAN, T. TUĞLULAR, and H. ŞENGONCA. Does Network Awareness Make Difference. ICHIT, 2006.

[13] Y. Breitbart, M. Garofalakis, B. Jai, C. Martin, R. Rastogi, and A. Silberschatz. Topology discovery in heterogeneous IP networks: the NetInventory system. *Networking, IEEE/ACM Transactions on*, 12(3):401–414, 2004.

[14] B. Lowekamp, D. O'Hallaron, and T. Gross. Topology discovery for large ethernet networks. *ACM SIGCOMM Computer Communication Review*, 31(4):237–248, 2001.

[15] D. Emma, A. Pescapé, and G. Ventre. Discovering topologies at router level. *Operations and Management in IP-Based Networks*, pages 118–129, 2005.

[16] The University of California. Jpcap Tutorial. (http://netresearch.ics.uci.edu/ kfujii/Jpcap/doc/tutorial/index.html), October 2010.

[17] Jason Crampton. XACML and Role-Based Access Control. (http://dimacs.rutgers.edu/Workshops/Commerce/slides/crampton.pdf), October 2010.

APPENDIX A

SAMPLE PACKET SENDER

A.1. Sample ICMP Packet Sender

```
NetworkInterface[] devices = JpcapCaptor.getDeviceList();
        if(args.length<1){
            System.out.print("Usage:java SentICMP");
            for(int i=0;i<devices.length;i++)
                System.out.println(i+":"+devices[i].name
                    +"("+devices[i].description+")");
            System.exit(0);
        }
        int index=Integer.parseInt(args[0]);
        JpcapSender sender=JpcapSender.openDevice(devices[index]);

        ICMPPacket p=new ICMPPacket();
        p.type=ICMPPacket.ICMP_TSTAMP;
        p.seq=1000;
        p.id=999;
        p.orig_timestamp=123;
        p.trans_timestamp=456;
        p.recv_timestamp=789;
        p.setIPv4Parameter(0,false,false,false,0,false,false,
                    false,0,1010101,100,IPPacket.IPPROTO_ICMP,
            InetAddress.getByName("www.yahoo.com"),
                    InetAddress.getByName("www.amazon.com"));
        p.data="data".getBytes();

        EthernetPacket ether=new EthernetPacket();
        ether.frametype=EthernetPacket.ETHERTYPE_IP;
        ether.src_mac=new byte[]{(byte)0,(byte)1,
                            (byte)2,(byte)3,(byte)4,(byte)5};
        ether.dst_mac=new byte[]{(byte)0,(byte)6,
                            (byte)7,(byte)8,(byte)9,(byte)10};
        p.datalink=ether;

        //for(int i=0;i<10;i++)
            sender.sendPacket(p);
    }
```

A.2. Sample ARP Packet Sender

```
public static byte[] arp(InetAddress ip) throws IOException{
        //find network interface
        NetworkInterface[] devices=JpcapCaptor.getDeviceList();
        NetworkInterface device=null;

        for(NetworkInterface d:devices){
            for(NetworkInterfaceAddress addr:d.addresses){
                if(!(addr.address instanceof Inet4Address))
                    continue;
                byte[] bip=ip.getAddress();
                byte[] subnet=addr.subnet.getAddress();
                byte[] bif=addr.address.getAddress();
                for(int i=0;i<4;i++){
                    bip[i]=(byte)(bip[i]&subnet[i]);
                    bif[i]=(byte)(bif[i]&subnet[i]);
                }
                if(Arrays.equals(bip,bif)){
                    device=d;
                    break loop;
                }
            }
        }

        if(device==null)
            throw new IllegalArgumentException(
                    ip+" is not a local address");

        //open Jpcap
        JpcapCaptor captor=
            JpcapCaptor.openDevice(device,2000,false,3000);
        captor.setFilter("arp",true);
        JpcapSender sender=captor.getJpcapSenderInstance();

        InetAddress srcip=null;
        for(NetworkInterfaceAddress addr:device.addresses)
            if(addr.address instanceof Inet4Address){
                srcip=addr.address;
                break;
            }

        byte[] broadcast=new byte[]{(byte)255,(byte)255,
                    (byte)255,(byte)255,(byte)255,(byte)255};
        ARPPacket arp=new ARPPacket();
        arp.hardtype=ARPPacket.HARDTYPE_ETHER;
        arp.prototype=ARPPacket.PROTOTYPE_IP;
        arp.operation=ARPPacket.ARP_REQUEST;
        arp.hlen=6;
        arp.plen=4;
        arp.sender_hardaddr=device.mac_address;
        arp.sender_protoaddr=srcip.getAddress();
        arp.target_hardaddr=broadcast;
        arp.target_protoaddr=ip.getAddress();

        EthernetPacket ether=new EthernetPacket();
        ether.frametype=EthernetPacket.ETHERTYPE_ARP;
        ether.src_mac=device.mac_address;
        ether.dst_mac=broadcast;
        arp.datalink=ether;

        sender.sendPacket(arp);

        while(true){
            ARPPacket p=(ARPPacket)captor.getPacket();
            if(p==null){
                throw new IllegalArgumentException(ip
                +" is not a local address");
            }
            if(Arrays.equals(p.target_protoaddr,
                            srcip.getAddress())){
                return p.sender_hardaddr;
            }
        }
    }

    public static void main(String[] args) throws Exception{
        if(args.length<1){
            System.out.println("Usage: java ARP <ip address>");
        }else{
            byte[] mac=ARP.arp(InetAddress.getByName(args[0]));
            for (byte b : mac)
                System.out.print(Integer.toHexString(b&0xff) + ":");
            System.out.println();
            System.exit(0);
        }
    }
```

APPENDIX B

THE RULE GENERATION MODULE TEST AND RESULT FILES

B.1. Global Policy File

```
<Policy>
      <id>policy1</id>
      <action></action>
      <protocol></protocol>
      <type>Policy</type>
      <children>
        <Policy>
          <id>rule1</id>
          <srcIP>any</srcIP>
          <srcPort>any</srcPort>
          <dstIP>any</dstIP>
          <dstPort>80</dstPort>
          <action>deny</action>
          <protocol>http</protocol>
          <type>Rule</type>
        </Policy>
        <Policy>
          <id>rule2</id>
          <srcIP>any</srcIP>
          <srcPort>any</srcPort>
          <dstIP>any</dstIP>
          <dstPort>8080</dstPort>
          <action>accept</action>
          <protocol>http</protocol>
          <type>Rule</type>
        </Policy>
      </children>
</Policy>
```

B.2. Local Policy File

```
<Policy>
      <id>policy1</id>
      <action></action>
      <protocol></protocol>
      <type>Policy</type>
      <children>
        <Policy>
          <id>rule1</id>
          <srcIP>any</srcIP>
          <srcPort>any</srcPort>
          <dstIP>any</dstIP>
          <dstPort>80</dstPort>
          <action>accept</action>
          <protocol>http</protocol>
          <type>Rule</type>
        </Policy>
      </children>
</Policy>
```

B.3. Sample Generated Rules File

```
<Rule RuleId="1" Effect="deny">
<Description>tcp,any,any,10.1.155.88,80,http</Description>
<Target>
<Subjects>
<Subject>
<SubjectMatch MatchId=
"urn:oasis:names:tc:xacml:2.0:function:ipAddress-regexp-match">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
^*:(any)$</AttributeValue>
<SubjectAttributeDesignator SubjectCategory=
"urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
 AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
 DataType="urn:oasis:names:tc:xacml:2.0:data-type:ipAddress" />
</SubjectMatch>
</Subject>
</Subjects>

<Resources>
<Resource>
```

```
<ResourceMatch MatchId=
"urn:oasis:names:tc:xacml:2.0:function:ipAddress-regexp-match">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
^(10)\.(1)\.(155)\.(88)\:(80)$</AttributeValue>

<ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:
1.0:resource:resource-id"
DataType="urn:oasis:names:tc:xacml:2.0:data-type:ipAddress" />

</ResourceMatch>
</Resource>
</Resources>

</Target>

<Condition>
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">

<Apply FunctionId=
"urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
<SubjectAttributeDesignator AttributeId="protocol"
DataType="http://www.w3.org/2001/XMLSchema#string" />

</Apply>

<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
tcp</AttributeValue>
</Apply>
</Condition>

</Rule>

<Rule RuleId="67" Effect="accept">
<Description>tcp,any,any,192.168.2.3,80,http</Description>
<Target>
<Subjects>
<Subject>
<SubjectMatch MatchId=
"urn:oasis:names:tc:xacml:2.0:function:ipAddress-regexp-match">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
^*:(any)$</AttributeValue>
<SubjectAttributeDesignator SubjectCategory="urn:oasis:names:tc:xacml:
1.0:subject-category:access-subject"
AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="urn:oasis:names:tc:xacml:2.0:data-type:ipAddress" />
</SubjectMatch>
</Subject>
```

```
</Subjects>

<Resources>
<Resource>
<ResourceMatch MatchId=
"urn:oasis:names:tc:xacml:2.0:function:ipAddress-regexp-match">

<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
^(192)\.(168)\.(2)\.(3)\:(80)$</AttributeValue>

<ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:
1.0:resource:resource-id"
DataType="urn:oasis:names:tc:xacml:2.0:data-type:ipAddress" />

</ResourceMatch>
</Resource>
</Resources>

</Target>

<Condition>
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
<Apply FunctionId=
"urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
<SubjectAttributeDesignator AttributeId="protocol"
DataType="http://www.w3.org/2001/XMLSchema#string" />

</Apply>

<AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">tcp</AttributeValue>
</Apply>

</Condition>

</Rule>
```