NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

VALIDATING NETWORK SECURITY POLICIES VIA STATIC ANALYSIS OF ROUTER ACL CONFIGURATION

by

Eric Gregory Wen Wei Wong

December 2006

Thesis Advisor:	Geoffrey Xie
Second Reader:	John Gibson

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** December 2006	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Validating Network Security Policies via Static Analysis of Router ACL Configuration		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Eric Gregory Wen Wei Wong			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER** N/A	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE**	

13. ABSTRACT (maximum 200 words)

The security of a network depends on how its design fulfils the organization's security policy. One aspect of security is reachability: whether two hosts can communicate. Network designers and operators face a very difficult problem in verifying the reachability of a network, because of the lack of automated tools, and calculations by hand are impractical because of the often sheer size of networks.

The reachability of a network is influenced by packet filters, routing protocols, and packet transformations. A general framework for calculating the joint effect of these three factors was published recently. This thesis partially validates that framework through a detailed Java implementation, with the creation of an automated solution which demonstrates that the effect of statically configured packet filters on the reachability upper bounds of a network can be computed efficiently. The automated solution performs its computations purely based on the data obtained from parsing router configuration files. Mapping all packet filter rules into a data structure called PacketSet, consisting of tuples of permitted ranges of packet header fields, is the key to easy manipulation of the data obtained from the router configuration files. This novel approach facilitates the validation of the security policies of very large networks, which was previously not possible, and paves the way for a complete automated solution for static analysis of network reachability.

14. SUBJECT TERMS Static Reachability, Network Analysis, Router Configuration, Network Security Policy			**15. NUMBER OF PAGES** 169
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

VALIDATING NETWORK SECURITY POLICIES VIA STATIC ANALYSIS OF ROUTER ACL CONFIGURATION

Eric Gregory Wen Wei Wong
Major, Singapore Army
Bachelor of Engineering (Hons) (Electrical and Electronic Engineering), Nanyang Technological University, Singapore, 2001

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 2006

Author: Eric Gregory Wen Wei Wong

Approved by: Geoffrey Xie
Thesis Advisor

John Gibson
Second Reader

Peter J. Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The security of a network depends on how its design fulfils the organization's security policy. One aspect of security is reachability: whether two hosts can communicate. Network designers and operators face a very difficult problem in verifying the reachability of a network, because of the lack of automated tools, and calculations by hand are impractical because of the often sheer size of networks.

The reachability of a network is influenced by packet filters, routing protocols, and packet transformations. A general framework for calculating the joint effect of these three factors was published recently. This thesis partially validates that framework through a detailed Java implementation, with the creation of an automated solution which demonstrates that the effect of statically configured packet filters on the reachability upper bounds of a network can be computed efficiently. The automated solution performs its computations purely based on the data obtained from parsing router configuration files. Mapping all packet filter rules into a data structure called PacketSet, consisting of tuples of permitted ranges of packet header fields, is the key to easy manipulation of the data obtained from the router configuration files. This novel approach facilitates the validation of the security policies of very large networks, which was previously not possible, and paves the way for a complete automated solution for static analysis of network reachability.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

This thesis is dedicated to my wife, Karyn. During the entire time of my study at NPS, she always gave her kind understanding, tireless support, and constant encouragement. She had been my motivation in giving of my fullest to this thesis effort. She had also imparted her considerable software debugging skills to me, which saw me through the difficult times of code writing and debugging.

I would also like to express my sincere gratitude to Professor Geoffrey Xie, for giving me the opportunity to work in an area which I had never been exposed to, for trusting that I would be able to deliver results in uncharted territory. His high standards challenged me to put in more effort as time passed, so that I could live up to his expectations. His invaluable guidance provided me with a deeper understanding of networking.

In addition, my appreciation goes out to Professor John Gibson, for his helpful input and candid feedback.

Special mention goes to Carnegie-Mellon University, for kindly sharing their router configuration files with us, so that we could develop the software with real-world input.

Last but not least, I would like to thank the Government of Singapore, for giving me the opportunity to study in this wonderful place called the Naval Postgraduate School.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Computer networks [1] today carry many different types of data, and support a myriad of applications such as teleconferencing, streaming audio and video, electronic commerce, electronic mail, the World Wide Web, instant messaging, file sharing, distributed computing, and digital libraries, just to name a few. The backbone of a network [2] consists of a collection of packet switches (commonly known as routers) and links. A link can either be of the point-to-point (e.g., direct connection between two nodes), or the multiple-point access (e.g., Ethernet, packet radio) variety.

There are different interest groups concerned with a network: the common ones are the network provider (or manager), network designer, network operator, application programmer, and user. The network provider is usually responsible for administration and management of the network, as well as specifying a security policy that protects the integrity of the network. The network designer would build the network to support user requirements, ensure that network resources are utilized efficiently, and fulfill the security policy requirements.

Network design is driven by policy. The design has to fulfill a given set of objectives and constraints. In order to achieve these objectives and constraints, network designers have a vast range of options available to them. Practically, it involves the specification of packet filters, routing policies, and link weights, among others. There are many ways to design a network that fulfils the policy-specified objectives and constraints.

Knowing the reachability of a network is an important and complex practical problem that affects network designers and policy makers of an organization. It is crucial that network programmers accurately build and maintain a network that fulfils the security policy of the organization. Based on information from networking research literature, not many efforts have been made in the area of reachability analysis.

It is a daunting task to try to determine host level reachability, i.e., what kinds of packets can be exchanged between two hosts in a network. Two hosts may be able to exchange packets because it was intentionally designed as such, or it could be due to a mistake (configuration or design). Knowing whether packets can be exchanged between two hosts in a network is crucial to building a secure network.

One common technique for determining reachability is to use tools (e.g., *traceroute, ping*) that experimentally test the reachability between two hosts. Manual static analysis, which involves calculating the static reachability of a network by hand (either through the use of the above-mentioned tools' results, or by poring over a network's design), is usually impractical because of the often sheer size of the network, in terms of the number of routers it contains. How to automate static reachability analysis is the focus of this thesis.

A. OBJECTIVE

Earlier research into this area has yielded a manual technique that defines the potential reachability of a network and computes the transitive closure of set union and intersection operations on reachability set representations. It is a very tedious process to apply this technique to large networks, and/or multiple networks. Thus there is an impetus to develop an automated solution that can be used to analyze networks of any size.

This research will delve into the implementation of the automated static analysis of a network, adding on to the work already done in developing both a tractable framework [3] for reasoning about a network's reachability, and the algorithms for computing reachability bounds. There is no known solution available that is capable of determining the static reachability of a network from just parsing the router configuration files, and hence verifying that the security policy is enforced. It is very difficult to verify that the security policy holds, given that router configuration changes occur rather often.

In this thesis, efforts were taken to automate the process of static reachability analysis in order for quick and easy verification of a network's

security policy. The static analysis will make use of a snapshot of a network's router configuration files and as such can be considered a white-box approach[1]. Static analysis is performed in particular on the access control lists (ACLs) and interface configurations stored within these router configuration files. Suitable data structures and algorithms must be created for this purpose.

The thesis aims to develop an automated solution that will, in many cases, be able to validate a network's security policies through static analysis of its configuration. Based on current knowledge of efforts in this field, such an undertaking has not been done on a comparable level, and this will be a significant contribution of this thesis. This research will pave the way for future attempts to address dynamic network states that will impact the reachability problem, and to perform "what-if" analysis of failure scenarios.

B. WHY STATIC ANALYSIS

IP networks are the backbone of the Internet and are also used in many private enterprise networks. Networking professionals are not only concerned with the physical topology of these networks (comprising routers and the links between them), but must also deal with the nontrivial task of configuring the mechanisms that affect communications between hosts. These mechanisms refer to the packet filters, routing protocols and packet transformations that are handled by the routers on the IP networks. Consequently, the reachability of a network is affected by the above-mentioned mechanisms [3].

Packet filters represent the easiest way to control reachability, and they are widely used by network designers and operators to control the flow of wanted or unwanted packets in the data plane. There is a wide variety of options that can be configured in each packet filter, e.g.,, IP addresses, ports used, protocols used. As a result of the many ways in which packet filters can be configured, it is easy for network operators to misconfigure them. Misconfiguration of packet filters can also be attributed to the difficulty in managing packet filters across an

[1] A white-box approach means the person working on the problem has a complete view (internal and external) of the system being worked upon, vis-à-vis a black-box approach where only the external system view is known.

entire network, especially large ones. It is difficult to envisage the impact and logic of packet filters across the network without a proper tool that can perform such analysis across the entire network. As a result, it becomes hard for network operators to say with absolute confidence that their network configuration satisfies the security policy in its entirety.

Routing protocols affect the paths that packets will traverse across a network. A router can learn routes either by having information on directly connected links, being configured with static routes, or dynamically exchanging routing information with other routers. The dynamic exchange of routing information is governed through the use of routing protocols. All these influence reachability because they affect the entries that make up a router's forwarding table.

Another aspect of reachability is influenced by packet transformations. The header fields that affect the decisions that routers make with respect to packet filtering and forwarding of each packet may change as the packet traverses the network.

Packet filters are static, in the sense that they remain relatively unchanged, and can usually be changed only by a network operator, as opposed to dynamic changes caused by fluctuations in the network.

Thus it is a natural step to first begin with the development of a tool that can automatically perform static analysis of a network based on packet filters, since packet filter use is so prevalent, and has a large impact on the reachability of a network.

Router configuration files, which are stored in each router, contain information on the complete set of router commands at a particular moment in time. These files contain packet filter configurations and interface configurations, among other information. Hence, white-box analysis can be performed on these router configuration files.

C. RESEARCH QUESTIONS

What would be a suitable automated solution to determine the static reachability of a network? Subordinate issues include:

a. What is the framework for computing reachability bounds?

b. How can the above framework be applied to a static description of a deployed network?

c. What is a suitable data structure for representing standard and extended ACLs that enables rapid analysis to be carried out?

d. How can a complex network model be reduced into a simple one for analysis?

e. How can the technique be used for computing finer-grain reachability bounds (e.g., consider only a subset of packets)?

D. ORGANIZATION

The organization of the thesis is as follows: Chapter II provides an overview of the reachability problem in IP networks, and a discussion of related research that includes industry efforts. Chapter III covers the approach taken in finding a solution to the static part of the reachability problem, and explores the techniques developed. Chapter IV outlines the implementation process, and highlights measures taken to ensure the success of the solution. It also takes a look at the results of the system operating with real-world data. Chapter V synopsizes this thesis effort and provides recommendations for follow-up research.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

This chapter considers the implications of the use of the Cisco Internetwork Operating System (IOS), which is the software controlling Cisco network devices. Cisco has been the clear market leader in routers and switchers for many years, and continues to dominate the market, over industry rivals such as Juniper Networks. In 2006, Cisco held 75% of the market share in routers, and 71% of the market share in switches [4]. Hence it makes sense to focus our attention on IOS compatible router configuration files.

After examining the scope of the problem, a short discussion of related research and industry efforts follows. Many network researchers have realized the importance of packet filters, and have dedicated efforts towards developing better classification algorithms, or improved filtering algorithms, among other ideas.

A. CISCO IOS

There are many different versions of the Cisco IOS. As of November 2006, the version number is Release 12.4. There are many features provided by the Cisco IOS, but the functionalities that this research will be referring to are the interface and ACL configuration aspects. This section will detail the inner workings of the Cisco IOS with respect to router configuration files.

1. Router Hostname

Each router configuration file will generally contain a keyword *hostname*. The word after the *hostname* keyword specifies the router name which should be unique per router.

`hostname` *`router-name`*

Example. hostname router.gw

The above example denotes a router with the name of router.gw.

2. Interface Configuration

The interface section begins with the keyword *interface*. Each interface on the router will have its own section, beginning with the *interface*

keyword. There is no keyword to denote the end of each interface section, but another keyword denotes the start of another section, e.g., *router* which denotes the start of a routing protocol section.

a. Interface Name

The word following the *interface* keyword is the interface name. That line is usually followed by a description of the interface, which begins with the keyword *description*. The syntax is as follows:

`interface` *`interface-name`*

Example. interface Vlan1

The above example denotes an interface known as Vlan1.

Example. interface Ethernet3/1

The above example refers to the first interface on the Ethernet device in slot 3 (the fourth slot on the router).

b. Interface IP Address

The ip address and subnet mask of the interface follows on another line, which begins with *ip address*, the syntax of which is as follows:

`ip address` *`ip-address subnet-mask`*

Example. ip address 120.2.1.0 255.255.255.192

The above example denotes an interface with the IP address of 120.2.1.0, and a subnet mask of 255.255.255.192. This means that other routers with interface IP address of 120.2.1.1, 120.2.1.2 through 120.2.1.63 are on the same subnet, and hence, are its neighbors.

c. ACL Activation

Under each interface, one or more ACL can be activated for packet filtering. In the absence of ACLs, all packets will be allowed. An ACL can be applied on the inbound or outbound queue. Such specification begins with *ip access-group.* The syntax is as follows:

ip access-group *access-list-number* {in|out}

Example. ip access-group 101 in

The above example denotes an interface filtering on the incoming interface, using ACL 101, which is specified later in the router configuration file. If no ACLs are activated on an interface, all packets are permitted through it.

3. Access Control List Definition

ACL entries are processed sequentially in the order that they are stored. Packets are compared against the first entry, then against the second entry if it doesn't match the first, and so on. The search continues until a match is reached, after which the subsequent ACL rules are not processed. If none of the rules are a match, the traffic is denied, because there is an implicit "deny all" rule. Hence, the order of the ACL entries are a very important consideration as each rule is either an explicit deny or permit action. Cisco IOS permits different types of ACLs [5], as shown below:

a. Standard ACLs

Standard ACLs were introduced in Cisco IOS Software Release 8.3. They filter traffic based on the source address in IP packets, in comparison to the source address configured in the ACL. The syntax [6] of the standard ACL is as follows:

access-list *access-list-number* {**deny** | **permit**} *source* [*source-wildcard*] [**log**]

Example. access-list 1 permit 100.10.1.0 0.0.0.255

The above example shows a standard ACL that permits traffic from source 100.10.1.x.

The access-list number can be any number from 1 to 99. The numbers 1300 to 1399 were added to the standard ACL range in Cisco IOS Software Release 12.0.1.

A *source/source-wildcard* setting of 0.0.0.0/255.255.255.255 can be specified as **any**. If the wildcard is all zeros (0.0.0.0), it can be omitted.

host 100.10.1.1 0.0.0.0

is the same as

host 100.10.1.1

b. Extended ACLs

Extended ACLs were also introduced in Cisco IOS Software Release 8.3. They are an extension of the standard ACLs, and filter traffic based on the source and destination addresses in IP packets, in comparison to that configured in the ACL. The syntax [6] for the extended ACLs are shown below.

The general syntax for IP packets is as follows:

access-list *access-list-number* [**dynamic** *dynamic-name* [**timeout** *minutes*]] {deny | permit} *protocol source source-wildcard destination destination-wildcard* [precedence *precedence*] [tos *tos*] [log | log-input] [time-range *time-range-name*] [**fragments**]

For Internet Control Message Protocol (ICMP), the syntax is:

access-list *access-list-number* [**dynamic** *dynamic-name* [**timeout** *minutes*]] {deny | permit} **icmp** *source source-wildcard destination destination-wildcard* [*icmp-type* | [[*icmp-type icmp-code*] | [*icmp-message*]] [**precedence** *precedence*] [**tos** *tos*] [log | log-input] [time-range *time-range-name*] [**fragments**]

For Internet Group Message Protocol (IGMP), the syntax is:

access-list *access-list-number* [**dynamic** *dynamic-name* [**timeout** *minutes*]] {deny | permit} **igmp** *source source-wildcard destination destination-wildcard* [*igmp-type*] [**precedence** *precedence*] [**tos** *tos*] [log | log-input] [time-range *time-range-name*] [**fragments**]

For Transmission Control Protocol (TCP), the syntax is:

access-list *access-list-number* [**dynamic** *dynamic-name* [**timeout** *minutes*]] {deny | permit} **tcp** *source source-wildcard* [*operator* [*port*]] *destination destination-wildcard* [*operator* [*port*]] [**established**] [**precedence** *precedence*] [**tos** *tos*] [log | log-input] [time-range *time-range-name*] [**fragments**]

For User Datagram Protocol (UDP), the syntax is:

access-list *access-list-number* [**dynamic** *dynamic-name* [**timeout** *minutes*]] {deny | permit} **udp** *source source-wildcard* [*operator* [*port*]] *destination destination-wildcard* [*operator* [*port*]] [**precedence** *precedence*] [**tos** *tos*] [log | log-input] [time-range *time-range-name*] [**fragments**]

The extended access-list number can be any number from 101 to 199. The numbers 2000 to 2699 were added to the range in Cisco IOS Software Release 12.0.1.

c. Lock and Key (Dynamic ACLs)

Lock and Key ACLs were introduced in Cisco IOS Software Release 11.1. They are dependent on Telnet, authentication (local or remote) and extended ACLs. To use Lock and Key ACLs, the extended ACL must first block traffic through the router. Users of the router must Telnet to the router and authenticate themselves. After authentication, the router adds a single-entry dynamic ACL to the extended ACL, which permits traffic for a specific time period. It is possible to define idle and absolute timeouts. The extended ACL must contain the keyword *dynamic*.

d. IP Named ACLs

The feature of using names instead of numbers (access-list-number) for standard and extended ACLs was added in Cisco IOS Software Release 11.2. The syntax for IP named ACLs is as follows:

ip access-list {extended | standard} *name*

e. Reflexive ACLs

Reflexive ACLs were introduced in Cisco IOS Software Release 11.3. Reflexive ACLs allow IP packets to be filtered based on upper-layer session information. They are generally used to allow outbound traffic and to limit inbound traffic in response to sessions that originate inside the router.

Reflexive ACLs can be defined only with extended named IP ACLs. They cannot be defined with numbered or standard named IP ACLs, or with other protocol ACLs. However, reflexive ACLs can be used in conjunction with other standard and static extended ACLs.

f. Time-Based ACLs

Time-based ACLs were introduced in Cisco IOS Software Release 12.0.1.T. They give users flexibility in access control in the time dimension. Time restrictions are imposed on the time function, and rely on the router system clock, although Cisco states that the use of Network Time Protocol synchronization would be best. This feature is invoked in extended ACLs by specifying the `[time-range` *`time-range-name`*`]` option.

g. Commented IP ACL Entries

Commented IP ACL entries were introduced in Cisco IOS Software Release 12.0.2.T. With comments, ACLs become easier to understand, just like the use of comments in regular programming languages. Commented IP ACL entries can be used in standard and extended ACLs. The syntax for comments is:

`ip access-list` `{standard|extended}` *`name`* `remark` *`remark`*

for named ACLs, and the following is for regular numbered ACLs:

`access-list` *`access-list-number`* `remark` *`remark`*

h. Other Forms of Access Control

There are also other forms of access control, albeit more specialized in nature. There is the **Context-based Access Control (CBAC)** which inspects all traffic and maintains state information for TCP and UDP sessions. **Authentication proxies** can be set up in firewalls to enable

authentication for inbound and/or outbound users. **Turbo ACLs** can be found in high-end routers, for the purpose of improving router performance, by processing ACLs more efficiently. There are also **Distributed Time ACLs**, which allow time-based ACLs on Cisco VPN-enabled 7500 series routers. **Receive ACLs** improve security on Cisco 12000 series routers. **Infrastructure Protection ACLs** are used to protect infrastructure equipment by permitting only authorized traffic to them (permitting all transit traffic too). **Transit ACLs** improve network security by explicitly permitting only necessary traffic into the network.

4. Miscellaneous

More information on how to configure and use ACLs can be found at the Cisco website [5-7].

B. RELATED RESEARCH

There has been a significant amount of work carried out by networking researchers in relation to packet filtering. However, little work seems to have been done on tackling the reachability problem, as there has been little literature available on the matter. Nonetheless, some of the results of the efforts on packet filtering are useful in the quest for a workable solution to the reachability problem. This section will outline some of the relevant research performed in the packet filtering arena.

1. Use of Network Security Tools

There are many different network security tools on the market that can perform network vulnerability testing. According to [8], some of the popular ones are Nessus, Snort, Metasploit framework, ping, and traceroute, among others. The problem with these tools is that they require a live network on which to work. Hence, they are not useful as a design tool. Furthermore, they operate at a low level, and are not able to provide network designers/operators with a high-level view of the network's security.

2. Detecting and Resolving Packet Filter Conflicts

Reference [9] tackles the problem of filter conflict and resolution. The authors studied how filter conflicts arise, and proved that (then) existing conflict

resolution schemes based on filter ordering were not always effective. They developed a system based on adding new filters that could be applied in all cases.

The paper uses the concept of depicting a filter F as a **k-tuple** (F[1], F[2], …, F[k]), where each field F[i] is a prefix bit string (map) that denotes a range of addresses or field values. A filter that processes on only IP source addresses would be classified as a 1-tuple; a filter that processes on both IP source and destination addresses would be classified as a 2-tuple; and so on. Typically, filters are crafted to range from 1-tuple to 5-tuple. A 5-tuple filter would process on IP source and destination addresses, source and destination ports, as well as the packet's protocol field. This concept of representing filters is not new, and had already been explored in previous research. Nonetheless, the k-tuple representation of packet filters presents a convenient way of denoting these filters.

The approach towards solving filter conflicts is to **add resolve filters** for each pair of conflicting filters. Simply put, if two filters, F and G, are in conflict, then a new (resolve) filter H is introduced, where $H = F \cap G$. This idea is important, because having overlapping packet filters not only creates problems with respect to satisfying policy, but also in terms of efficiency, in having to process the same area multiple times.

3. Access Control List Analysis (ACLA)

In the ACLA framework [10], ACL analysis was automated, providing functionalities such as allowing detection and removal of redundant rules, discovery and repair of inconsistent rules, as well as merging of overlapping/adjacent rules. It also enabled translating a complex ACL (of both permits and denies) into a form consisting of only either permits or denies. The ACLA framework was topped off with a feature that could compute a meta-ACL profile based on all ACLs along a network path.

A dynamic multidimensional interval tree-based approach was adopted, with four dimensions of a packet filter considered: source and destination

addresses, protocol and port number. The first two were ranges, while the latter two were a single number each. The ordering of the final ACL was similar to the start state.

The algorithms developed were an important contribution to the reachability studies, but the number of dimensions considered did not cover the entire functionality of a packet filter. It was not a totally automated process, as the clients still had to implement the relations contained in the ACLA library. The ACLA framework was also not able to automatically derive the global view of the network from its existing topology and router configurations, and verify whether the network satisfied the security policy. The concepts are bold and innovative, but are only able to address a meta-ACL profile based on ACLs along a single network path, which is not addressing the complete reachability problem, in which multiple (possibly complex) paths are available from source to destination.

4. Algorithms for Packet Classification

Networking researchers have developed many different search algorithms to aid in packet classification [11]. The importance of this matter lies in speed, because fast links require fast classification.

In a simple sense, a packet represents a point in a two-dimensional space, if only the source and destination addresses are considered. Multi-dimensional classification would inherently be more complicated, and the problem is complicated even more by the need to match packets on ranges as well as prefixes. A two-dimensional rule is represented by a rectangle, and a *d*-dimensional rule is represented by a *d*-dimensional rectangle. Thus the schemes that make use of rectangular representation depict rules using *N* rectangular regions. Since packets are not only matched based on prefixes, but also on ranges, the term *W* is used to denote the number of bits in a range.

The algorithms explored include basic data structures (e.g., linear search), geometry-based structures (e.g., grid-of-tries), also known as string search (retrieval) trees, heuristic-based (e.g., tuple-space search), and hardware-only (e.g., bitmap-intersection) types.

The **hierarchical trie** structure (basic data structure) appears promising for use in this thesis, as the query time complexity for d-dimensions is $O(W^d)$. This structure was also used in [9] for the optimized solution. This data structure is also used by AT&T in their network backbone. Although the **grid-of-tries** has a better query time complexity, it is more suited for two-dimensional problems. Many other algorithms are also geared toward two-dimensional applications, and those that work well with multi-dimensions require a much greater storage complexity.

A summary of the classification schemes presented in the paper [11] is shown below:

Algorithm	Worst-case time complexity	Worst-case storage complexity
Linear Search	N	N
Ternary CAM	1	N
Hierarchical Tries	W^d	NdW
Set-pruning Tries	dW	N^d
Grid-of-Tries	W^{d-1}	NdW
Cross-producting	dW	N^d
FIS-tree	$(l+1)\,W$	$l \times N^{1+1/l}$
RFC	d	N^d
Bitmap-intersection	$dW + N/memwidth$	dN^2
HiCuts	d	N^d
Tuple Space Search	N	N

Table 1. Summary of Classification Schemes (From [11])

5. Static Reachability Analysis of IP Networks

The static reachability analysis paper [3] presents a precise definition of the potential reachability of a network, accounting for the influences of packet filters, routing protocols, and packet transformations. In addition, a tractable framework for calculating the effects of the above-mentioned three factors on the reachability in a network was developed, and the algorithm is based on static analysis of router configuration files.

Packet filters are seen as the easiest way to control reachability. The filters contain clauses that permit or deny packets based on their header fields. Each interface on a router may be configured with different filters for inbound and outbound packets, and this ultimately affects reachability.

Routing protocols also control reachability because they affect the contents of a router's forwarding table. Each routing process (i.e., an instance of a routing protocol) controls the sharing of routing information between routers, and routing processes have to be specifically configured to share this routing information.

Packet transformations also affect reachability in a network because packet header fields may change as the packet traverses the network. Since routing is based on these header fields, any change in them would affect reachability.

Of particular interest, the reachability of a network has 2 key bounds: an upper bound and a lower bound. **Reachability Upper Bound** (RUB) refers to the set of packets that could potentially reach from nodes i to j in the network, if routing decisions were made appropriately, and is defined as follows (where s is the network's forwarding state, and S is the set of all possible network forwarding states) :

$$R_{i,j}^{U} = \bigcup_{s \in S} R_{i,j}(s)$$

Reachability Lower Bound (RLB) refers to packets that will be guaranteed delivery from i to j as long as the network is not physically partitioned, and is defined as follows:

$$R_{i,j}^{L} = \bigcap_{s \in S} R_{i,j}(s)$$

The RLB algorithm is based on the "Articulation Points and Biconnected Components" algorithm for any pair of nodes i and j, and subsequently determining the intersection of $F_{u,v}$ for the remaining edges. Note that $F_{u,v}$ refers to the set of packets that the network is able to carry over edge <u, v>. An alternate view of $F_{u,v}$ is a packet filter representation with different predicates.

As for the RUB, the algorithm was developed as follows (note that R denotes reachability between two points, and V is the set of routers):

```
1. Initialize R(i,j) to F_{i,j} for all i;
2. for (m = 1 to ||V|| - 2) do
3.     for (i = 1 to ||V||) do
4.         R'(i,j) = ∅;
5.         for (k = 1 to ||V||) do
6.             if (<i,k>∈E)
                   then R'(i,j) = R'(i,j) ∪ {F_{i,k} ∩
                   R(k,j)};
7.         R(i,j) = R'(i,j);
```

With the RLB and RUB algorithms, it becomes possible to calculate the static reachability of a network with ease, and it is a matter of inserting the effects of the three factors of packet filters, routing protocols and packet transformations into $F_{u,v}$.

C. CONCLUSION

After examining the syntax used by the Cisco IOS, it is now possible to develop a parser to extract the pertinent information from router configuration files that would be needed to perform reachability bound calculations. Although there are several pieces of related research, none are feasible for implementation in this effort, save for the use of a tuple-based data structure.

The algorithm developed to compute RUB is most applicable to this thesis, as the results would be essential in determining network reachability. All of these will be examined in greater detail in the next chapter for implementation purposes.

THIS PAGE INTENTIONALLY LEFT BLANK

III. DESIGN

This chapter explores the methodology and the actual design for the thesis work. The reachability bound computation revolves around the analysis of network router configuration files. A look will be taken at the data structure developed for the automated solution to work with, as well as the algorithms modified from previous research and new ones written as part of this effort. To complete the design, the human-computer interaction aspect will be studied to create an effective GUI front end to the analysis tool.

A. SET NOTATION OF PACKET FILTERS

1. PacketSet Data Structure

A **PacketSet** is essentially a representation of a packet filter (i.e., ACL) in a set form. It denotes the set of packets *permitted* by that packet filter. Each ACL rule is denoted by a 5-tuple, with the five dimensions as follow:

- Source IP address
- Destination IP address
- Source port number
- Destination port number
- Protocol number

In set notation, the 5-tuple would be:

`[src-ip];[src-port];[dest-ip];[dest-port];[prot]`

where :
- `src-ip` : source IP address
- `src-port` : source port
- `dest-ip` : destination IP address
- `dest-port` : destination port
- `prot` : protocol

Also, each dimension of the 5-tuple would actually be a range of values permitted. The PacketSet data structure would thus be:

```
[src-ip lower, src-ip upper] ; [src-port lower,
src-port upper] ; [dest-ip lower, dest-ip upper]
; [dest-port lower, dest-port upper] ; [prot
lower, prot upper]
```

The source and destination IP addresses have a maximum range from 0.0.0.0 to 255.255.255.255, the source and destination ports from 0 to 65535, and the protocol number from 0 to 255.

The **PacketSet** is based on a packet filter, yet there is subtle but important difference. Packet filters contain rules that must be processed sequentially, in the order they are stored in the list. The PacketSet data structure that was developed contains 'rules' (5-tuples) that **do not** have to be processed sequentially.

This relaxation of the sequential processing rule enables greater flexibility in optimizing, reordering and restructuring the 5-tuples. The secret as to how the PacketSet can afford to do away with sequential processing is that all permit and deny rules in a packet filter are mapped to permit-only 5-tuples. The mapping of packet filter rules to 5-tuples will be covered in the next section.

2. PacketSet Creation Algorithm

Creating a PacketSet is a complex task, as it requires the sequential handling of a jumble of permit and deny rules typically found in a packet filter. The mapping of permit and deny rules in a packet filter to tuples of a PacketSet is not one-to-one, because (i) deny rules by themselves do not cause tuples to be added, and (ii) a permit rule may add zero, one or more tuples, depending upon the preceding deny rules.

The algorithm that follows is responsible for converting one ACL into a PacketSet, and assumes that the packet filter rules in a router configuration file have already been parsed:

CreatePacketSet (*ACL*)

```
1.  Create empty PacketSet;
2.  Create an empty Deny Buffer;
3.  For (each rule in ACL), do
4.  Create an empty Interim Buffer;
5.  Convert rule into a new Current Tuple;
6.  If rule is “Deny”, add Current Tuple to Deny
    Buffer;
7.  Else if rule is “Permit”
8.  Add Current Tuple to Interim Buffer;
9.  For (j=0 to size of Deny Buffer), do
10. For (k=0 to size of Interim Buffer), do
11. Perform a GetPermitTuple on Interim Buffer[k]
    against Deny Buffer[j];
12. Add Interim Buffer to PacketSet;
13. If PacketSet is not empty, perform optimization
    to merge and remove overlapping ranges;
```

What the algorithm does is to sequentially go through a list of ACL rules, and save the permit rules it encounters into a buffer until it finds a deny rule. Every deny rule goes into the deny buffer. After the first deny rule is encountered, every subsequent permit rule is compared against the rules in the deny buffer to generate a new tuple, before adding it to the PacketSet. Finally, the PacketSet is saved as part of the ACL object..

The essence of the GetPermitTuple function will be explored in the next section.

3. Mapping of Permit and Deny Rules to a Tuple

One of the key ingredients of the PacketSet creation algorithm lies in the GetPermitTuple function, which performs a mapping of permit and deny rules to a tuple. It is only invoked when there is a need to compare a permit rule against a deny rule (which comes before the permit rule).

a. Definition of Terms

A 1-tuple is written as follows:

$$[\ (A,1)_{lower},\ (A,1)_{upper}\ ]$$

The A in (A,1) denotes which tuple it is, and the 1 denotes the dimension. The upper and lower denote whether the value is the upper or lower figure in the range.

b. 1-Tuple Case

In comparing a permit rule against a deny rule (which comes before the permit rule), it is best to start with the simplest case, the 1-tuple. The first 1-tuple will correspond to the deny rule, and the second will correspond to the permit rule (an example illustrates this below). Subsequent cases will also build upon this logic.

For:

$$[(A,1)_{lower},(A,1)_{upper}]$$

$$[(B,1)_{lower},(B,1)_{upper}]$$

where A is the deny tuple, and B is the permit tuple.

There are three possible outcomes to the comparison:

- No tuple is generated if B's range is a subset of A's range.
- One tuple is generated if only one end of both A and B overlap
- Two tuples are generated if A's is a subset of B's range (i.e., B's range extends beyond both bounds of A's range).

The result can be written succinctly in a mathematical notation as follows:

$$[(B,1)_{lower},\ \min\{(A,1)_{lower}-1,(B,1)_{upper}\}]\ \cup$$

$$[\max\{(B,1)_{lower},\ (A,1)_{upper}+1\},(B,1)_{upper}]$$

Note: each of which is valid only if the lower value $\leq$ upper value. Thus if any range is not valid, the entire row (tuple) is invalid, and should be rejected.

Example:

[3,5] → deny

[1,7] → permit

The result generated would be [1,2] ∪ [6,7].

c. 2-Tuple Case

The problem gets slightly more complicated with the addition of another dimension. However, the logic is still similar, as there is a pattern to be seen.

For:

$[(A,1)_{lower},(A,1)_{upper}];\ [(A,2)_{lower},(A,2)_{upper}]$

$[(B,1)_{lower},(B,1)_{upper}];\ [(B,2)_{lower},(B,2)_{upper}]$

The result of comparing the permit rule B against deny rule A would be:

$[(B,1)_{lower},\min\{(A,1)_{lower}-1,(B,1)_{upper}\}];$
$[(B,2)_{lower},(B,2)_{upper}] \cup$

$[\max\{(B,1)_{lower},(A,1)_{upper}+1\},(B,1)_{upper}];$
$[(B,2)_{lower},(B,2)_{upper}] \cup$

$[(B,1)_{lower},(B,1)_{upper}];$
$[(B,2)_{lower},\min\{(A,2)_{lower}-1,(B,2)_{upper}\}] \cup$

$[(B,1)_{lower},(B,1)_{upper}];$
$[\max\{(B,2)_{lower},(A,2)_{upper}+1\},(B,2)_{upper}]$

There is a pattern that can be seen forming.

Example:

[3,5];[10,12] → deny

[1,7];[9,13] → permit

The result generated would be:

$[1,2];[9,13] \cup$

$[6,7];[9,13] \cup$

$[1,7];[9,9] \cup$

$[1,7];[13,13]$

d. 3-Tuple Case

Again, like the 2-Tuple case to the 1-Tuple case, the 3-Tuple case is like an extension of the 2-Tuple case. Applying the same logic to:

$[(A,1)_{lower},(A,1)_{upper}];\ [(A,2)_{lower},(A,2)_{upper}];$
$[(A,3)_{lower},(A,3)_{upper}] \rightarrow$ deny

$[(B,1)_{lower},(B,1)_{upper}];\ [(B,2)_{lower},(B,2)_{upper}];$
$[(B,3)_{lower},(B,3)_{upper}] \rightarrow$ permit

The result of the comparison would be:

$[(B,1)_{lower},\min\{(A,1)_{lower}-1,(B,1)_{upper}\}];$
$[(B,2)_{lower},(B,2)_{upper}];[(B,3)_{lower},(B,3)_{upper}]$
$\cup$

$[\max\{(B,1)_{lower},(A,1)_{upper}+1\},(B,1)_{upper}];$
$[(B,2)_{lower},(B,2)_{upper}];[(B,3)_{lower},(B,3)_{upper}]$
$\cup$

$[(B,1)_{lower},(B,1)_{upper}];[(B,2)_{lower},$
$\min\{(A,2)_{lower}-1,(B,2)_{upper}\}];$
$[(B,3)_{lower},(B,3)_{upper}] \cup$

$[(B,1)_{lower},(B,1)_{upper}];$
$[\max\{(B,2)_{lower},(A,2)_{upper}+1\},(B,2)_{upper}];$
$[(B,3)_{lower},(B,3)_{upper}] \cup$

$[(B,1)_{lower},(B,1)_{upper}];[(B,2)_{lower},(B,2)_{upper}];$
$[(B,3)_{lower},\min\{(A,3)_{lower}-1,(B,3)_{upper}\}] \cup$

$[(B,1)_{lower},(B,1)_{upper}]$; $[(B,2)_{lower},(B,2)_{upper}]$;
$[max\{(B,3)_{lower},(A,3)_{upper}+1\},(B,3)_{upper}]$

The pattern is becoming more obvious. For every dimension, there is an addition of 2 tuples to the maximum number of tuples possibly generated. Hence, the total number of possible tuples for d dimensions is 2d.

From the above 3-Tuple case, and the 2-Tuple case, overlapping ranges can be seen. Thus optimization of the results can be carried out, in order to remove overlapping ranges.

Note that the pattern was:

[non-overlapping 1st-D permit range]; [2nd-D permit range]; [3rd-D permit range] ∪

[1st-D permit range]; [non-overlapping 2nd-D permit range]; [3rd-D permit range] ∪

[1st-D permit range]; [2nd-D permit range]; [non-overlapping 3rd-D permit range]

where non-overlapping refers to the range over which permit and deny tuples do not intersect

After optimization, the sequence becomes:

[non-overlapping 1st-D permit range]; [2nd-D permit range]; [3rd-D permit range] ∪

[overlapping 1st-D range]; [non-overlapping 2nd-D permit range]; [3rd-D permit range] ∪

[overlapping 1st-D range]; [overlapping 2nd-D range]; [non-overlapping 3rd-D permit range]

The overlapping 1st-D range (similarly for 2nd-D) can be represented by:

$$[\max\{(A,1)_{lower},(B,1)_{lower}\},\ \min\{(A,1)_{upper},(B,1)_{upper}\}]$$

Note: valid only if the lower value ≤ upper value. Thus if any range is not valid, the entire row (tuple) is invalid, and should be rejected.

e. 5-Tuple Case

With the emergence of a pattern in the above cases, it becomes simple to extend the logic to a 5-tuple case, or even to the x-tuple case.

[non-overlapping 1st-D permit range]; [2nd-D permit range]; … ; [xth-D permit range]

[overlapping 1st-D range]; [non-overlapping 2nd-D permit range]; [3rd-D permit range]; … ; [xth-D permit range]

[overlapping 1st-D range]; [overlapping 2nd-D range]; [non-overlapping 3rd-D permit range]; [4th-D permit range]; … ; [xth-D permit range]

The overlapping 1st-D range (similarly for 2nd-D) can be represented by:

$$[\max\{(A,1)_{lower},(B,1)_{lower}\},\ \min\{(A,1)_{upper},(B,1)_{upper}\}]$$

Note: valid only if the lower value ≤ upper value. Thus if any range is not valid, the entire row (tuple) is invalid, and should be rejected.

4. Importance of the PacketSet Data Structure

The development of the PacketSet is a very important step in this research. Most of the complication in the operations is taken away with the relaxation of the sequential processing rule. There is no longer a need to

process permit and deny rules separately. There is no longer a need to worry about the logic behind union and intersection operations (which are key to reachability computations) when having to deal with permit and deny rules.

The PacketSet is a powerful mechanism with which various forms of manipulation can be performed. The logic in maintaining a PacketSet data structure is greatly simplified vice a traditional packet filter structure. It is a streamlined form of representing a packet filter that enables set operations to be carried out easily. The PacketSet data structure is also extensible, in that more dimensions can be added when the need arises, e.g., for a more complete representation of a packet filter rule with the many options developed by Cisco. With a powerful and flexible data structure in place, the next logical step is to design the operations which are required to perform reachability computations.

B. OPERATIONS

1. Union Operation

The union operation is one of the two basic operations of the reachability computation [3]. As the complexity of manipulating permit and deny rules has been taken out of the equation with the development of the PacketSet data structure, the union operation becomes a relatively simple task to perform.

The algorithm for the union operation is as follows:

1. `Combine tuples of PacketSet 1 and PacketSet 2;`
2. `Perform optimization on the result of Step 1 to merge and remove overlapping ranges;`

2. Intersection Operation

The intersection operation is the other one of two basic operations of the reachability computation [3]. Similar to the union operation, because the complexity of manipulating permit and deny rules has been eliminated with the development of the PacketSet data structure, the intersection operation becomes a relatively simple task to perform.

The algorithm for the intersection operation is as follows:

1. Create an output PacketSet;
2. For each tuple in PacketSet 1, do
3. For each tuple in PacketSet 2, do
4 Check intersection of dimension/range 1 of the two PacketSets being compared;
5. Repeat Step 3 for dimensions/ranges 2 through 5;
6. If all five dimensions/ranges intersect, combine the intersection results of Steps 3 and 4 to give the output of the intersection operation;
7. Else output is an empty PacketSet;
8. Do a Union of the output obtained from Step 5 or 6 with the output PacketSet;
9. Perform optimization to merge and remove overlapping ranges in the output PacketSet;

3. Optimization Operation

The optimization mentioned in Section A, is the same as the optimization mentioned in the union and intersection operation sections. This operation aims to merge and remove overlapping ranges in the tuples.

The algorithm for the optimization operation is as follows (assuming a PacketSet called *input* is the target of the operation):

1. Do
2. Initialize Boolean toContinue = false;
3. For (i=0 to size of *input*), do
4. For (j=i+1 to size of *input*), do
5. Get tuple i from *input*;
6. Get tuple j from *input*;
7. If all ranges in tuples i and j overlap, do
8. Calculate the merged tuple;
9. Remove tuples i and j from *input*;
10. Add merged tuple at position i in *input*;
11. toContinue = true;

```
12.                 j--;
13.  While (toContinue);
```

Since the ordering of the ‘rules’ in a PacketSet does not matter, the above algorithm will compare every tuple in the PacketSet with every other tuple to try to determine whether there are overlapping areas and perform a merging of the two tuples. Merging will also take place for tuples with adjacent ranges.

C. NEIGHBOR COMPUTATION

In order for the reachability algorithm to traverse the network topology based solely on router configuration files, it was necessary to estimate the network topology by means of computing the neighbors of routers.

Routers on the same subnet are taken to be neighbors operating from the same switch. Thus, to compute a router’s neighbors, the IP address and mask on every interface of that router is used to determine its network prefix, and is compared against the network prefix of all interfaces on every other router. If they match, both interfaces on the two routers are taken to be neighbors.

D. REACHABILITY COMPUTATION

With the tools in place, it is now possible to perform RUB computations [3]. In the algorithm highlighted in Chapter II, the set of packets that the routers will permit are influenced by both packet filters and routing protocols, hence there might be differences in the packets permitted from a router (on each interface) to each of its neighbors. For example, although a packet filter on a router permits a particular packet to pass to all of its neighbors, a routing protocol might not route the same packet to some of the router’s neighbors. However, in this thesis, only the effects of packet filters are considered, hence the set of permitted packets from a router to its neighbors will be the same to every neighbor on each interface (assuming the out and in filters are the same on all nodes being considered).

The algorithm highlighted in Chapter II was adapted to work with the PacketSet data structure. The algorithm calculates the reachability from all routers to one single destination router, and extracts the reachability results for the user-defined source-destination pair of routers.

In order to compute $F_{i,j}$ (which refers to the set of packets that the network is able to carry over edge <i, j>, and where u corresponds to an interface on i, and v corresponds to an interface on j), the following algorithm was developed:

1. For (each edge <i,j> in network) do
2. Initialize $F_{i,j}$ to empty PacketSet;
3. For (each physical link <u,v>) do
4. Obtain all the ACLs activated by outbound queue of u;
5. PacketSet S1 = Intersection of all PacketSets specified by the ACLs obtained at step 4;
6. Obtain all the ACLs activated by inbound queue of v;
7. PacketSet S2 = Intersection of all PacketSets specified by the ACLs obtained at step 6;
8. $F_{i,j} = F_{i,j} \cup (S1 \cap S2)$;

Hence, the algorithm to compute the RUB for a given destination j is as follows:

```
1.  Initialize packetSetRUB[i][j] for all i:
      to F_{i,j} if i and j are neighbors;
      to whole PacketSet, if i=j;
      to empty PacketSet, otherwise;
2.  for ( m=0 to (numberOfRouters - 3) ) do
3.    for ( i=0 to (numberOfRouters - 1) ) do
4.      tempPacketSetRUB[i][j] = empty;
5.      for (each interface (z) on router i) do
6.        for ( k=0 to numberOfRouters - 1 ) do
7.          if (k has an interface that is a
              neighbor of i on interface z)
8.            intersectedPacketSetRUB = F_{i,k} ∩
                packetSetRUB[k][j];
```

```
9.                tempPacketSetRUB[i][j] =
                     tempPacketSetRUB[i][j] ∪
                     intersectedPacketSetRUB ;
10.            endif;
11.          endloop;
12.       packetSetRUB[i][j] = tempPacketSetRUB[i][j];
```

The algorithm will thus generate, for each <i,j> node pair, a PacketSet for the RUB.

E. USER GUI

The user Graphical User Interface (GUI) will enable users to navigate through the functions of the automated solution easily. It will allow user selection of the routers to be analyzed and subsequent presentation of the results of the analysis. The design of this GUI will be crucial in presenting the users with the right type of information in an easy-to-understand format.

1. Target Users

The expected users of the system are the network designers and operators. These users are trained professionals who are familiar with the network's design and router configuration programming.

a. User Characteristics

The primary user of the system is a network designer/operator, whose job is to ensure that network resources are utilized efficiently, and fulfill the security policy requirements. The system will be used for calculating the static reachability of a network, during the design phase or when verifying security policy compliance. It would be used occasionally, though not likely on a daily basis. It is difficult to predict the frequency of use, since it will be called upon only as need arises. It could range from once a week to once a month.

b. User Skills

Users are networking professionals, and hence would be familiar with basic computer operation and networking vernacular in general. They would probably have typing skills, though the system would not require it. Apart from

mouse clicks, a few keyboard strokes at the most, would be required. Being in the computer industry, users would be familiar with web browsers and programs in general.

2. Design Criteria

The design criteria of the user GUI encompass the following:

a. Easy to Learn

The user needs to be able to learn how to use the system with minimal training. The user is guided through step by step interfaces.

b. Minimum User Memory Load

The system is not expected to be used frequently, and the user should be able to maintain proficiency even with very sporadic usage.

c. Intuitive Presentation

The system should present information, such as progress status or error alerts, in a concise and intuitive manner.

d. Efficient

The user should not be overloaded with unnecessary information. Information must be concise and unambiguous for the user to carry out this task as quickly as possible, with minimal actions required for interaction.

e. Error Tolerant

The system should not allow users to enter incorrectly formatted data. It should show relevant and meaningful error messages.

f. Engaging

The tone and style of the interface should be pleasing to use.

3. Functionalities

The user GUI will feature several functions.

a. Parsing of Router Configuration Files

- User selection of input directory containing router configuration files.
- User selection of output directory where results will be saved.
- User initiated parsing of the router configuration files in the input directory.
- Parsed output would be saved to the output directory.

b. Reachability Computations

- User selection of two routers for reachability bound computation to be performed upon.
- User initiated reachability bound computation of the two selected routers.
- Reachability computations would be saved to the output directory specified earlier.

c. Miscellaneous

- Inform user of program errors, with descriptions where possible.
- Prevent erroneous data entry.

4. Conceptual Design

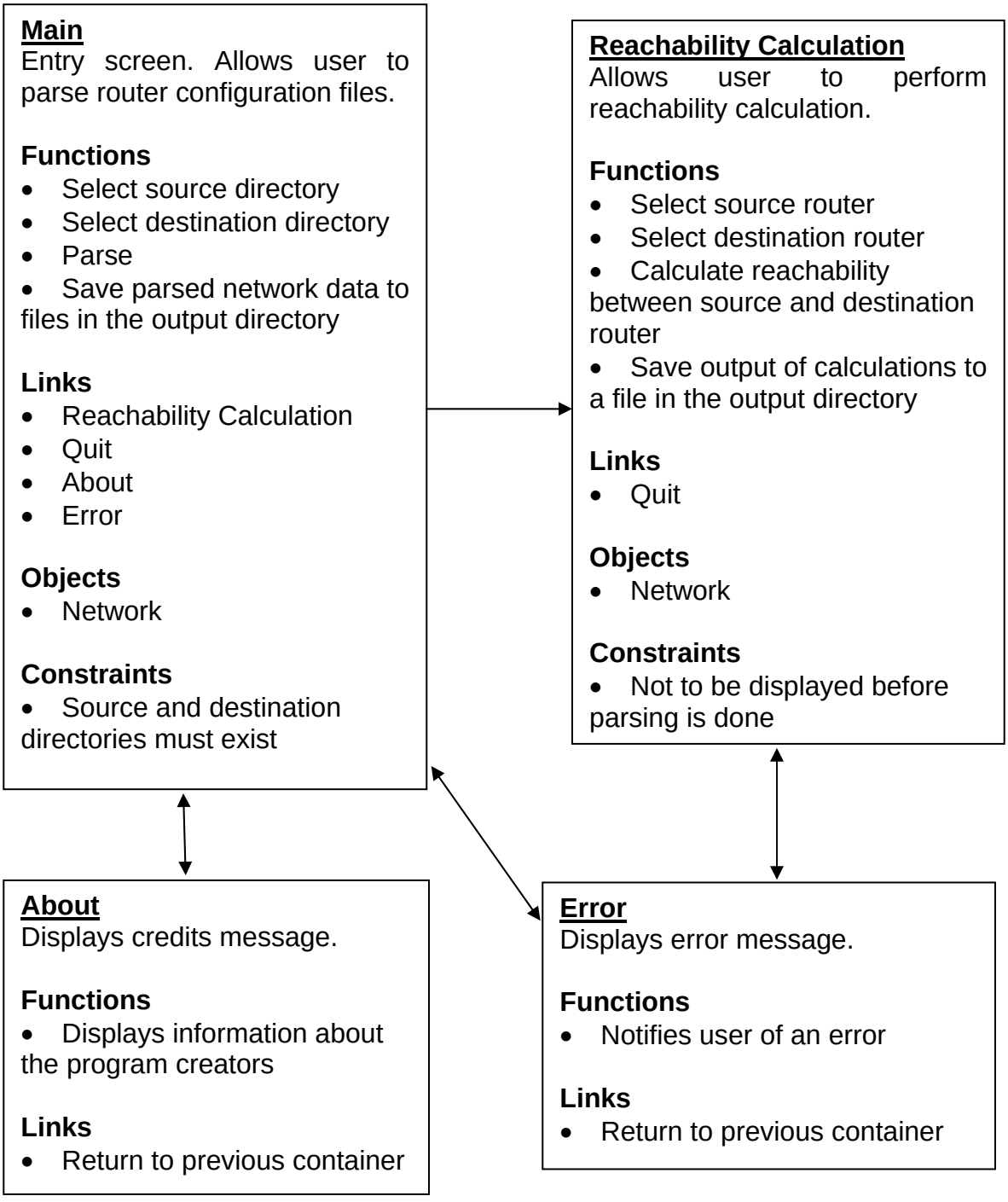

Figure 1. Content Diagram for the Static Reachability Analysis Toolkit

5. Visual Design

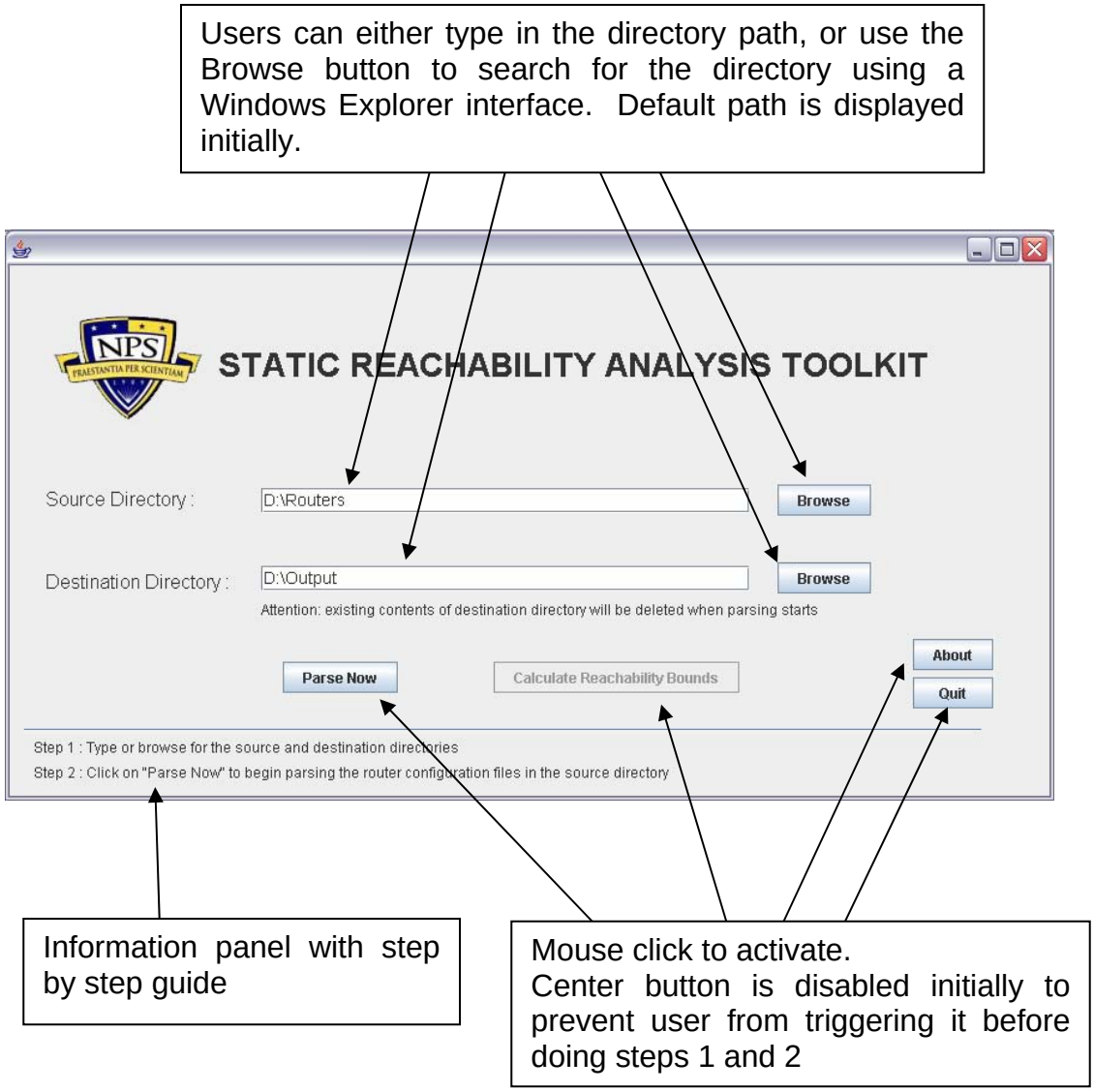

Figure 2. Main Screen

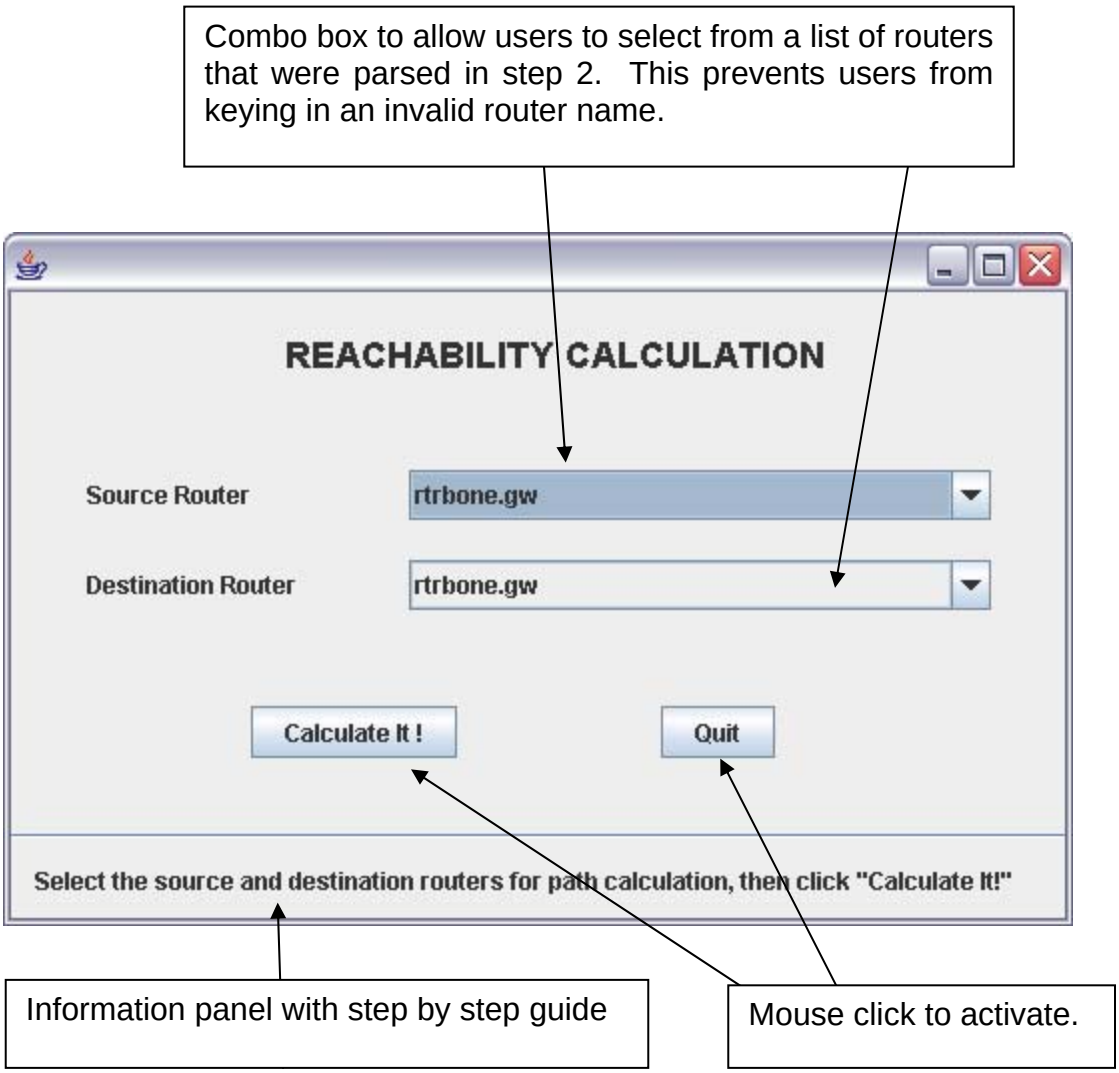

Figure 3. Reachability Calculation Screen

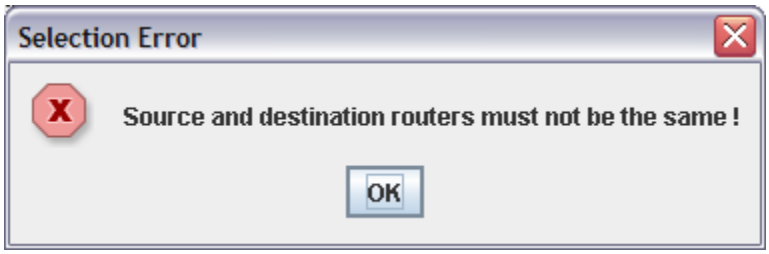

Figure 4. Sample Error Message Dialog Box

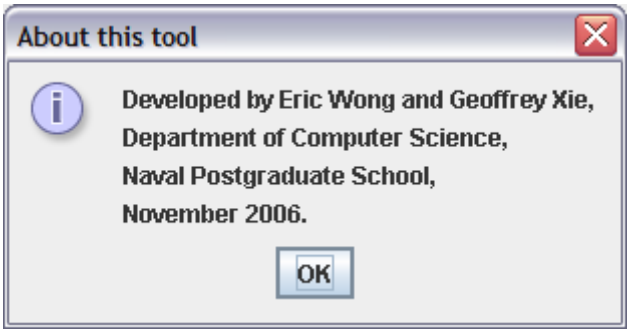

Figure 5. Information Dialog Box

F. CONCLUSION

In this chapter, the different aspects of the system had been designed, from the algorithms to the human-computer interface. One of the most important aspects of the design was the PacketSet data structure. It is the key to this research effort, because it greatly simplifies the union and intersection operations that the RUB algorithm relies upon. In the next chapter, the implementation of this design will be covered, with specific examples to illustrate an operational system.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. IMPLEMENTATION

This chapter describes the implementation of the design from Chapter III, with the creation of a Static Reachability Analysis Toolkit, using Java. The steps taken by the software are explored in detail. An example is provided to illustrate how the software works. Building upon that example, a security policy related example is given to show the usefulness of this Toolkit. To provide an overview of the entire software design, the class diagram would also be incorporated. The measures that were taken to ensure the success of the system are also highlighted. The creation of the Static Reachability Analysis Toolkit underscores the success of this research effort.

A. SOFTWARE IMPLEMENTATION

The system software was created with Java, using NetBeans IDE 5.0 [12]. NetBeans is a free, open-source IDE for software developers, and runs on many platforms such as Windows and Linux. The program does not require large amounts of CPU power or memory. Most of the research was performed on a Pentium M 1.86GHz CPU with 1GB of RAM. Java version 1.5.0_07-b03 was used to compile the code. The algorithms mentioned in Chapter III were implemented with Java.

1. Assumptions and Limitations

It was assumed that router configuration files followed the Cisco IOS syntax exactly, since the router configurations had to have been entered by a Cisco IOS interface.

Of the different types of ACLs mentioned in Chapter II, only numbered standard and extended ACLs are handled by the system (e.g., named ACLs are not handled).

2. Sample Router Configuration File

Router configuration files are snapshots of a router's configuration at a particular moment in time. The reachability analysis is based solely on these router configuration files. A sample router configuration file is attached in Appendix B.

3. Parsing a Router Configuration File

The first stage of the automated solution is the parsing of router configuration files. The Java program contains a parser that handles router configuration files stored in a directory, which a user can select. Upon completion of the parsing operation, the resulting files are stored in a user-selected directory.

The parsing operation is a two step function. The first step converts each router configuration file into a raw parsed format. The second step reads the ACLs from each parsed router configuration and generates the PacketSets for each ACL.

Two types of output files would be generated subsequently: a `Network Data.txt` file, and `Router dump – <`*`router name`*`>.txt` (where <router name> is the hostname stored in each router configuration file) files. The former contains the network id, as well as a list of routers processed. The latter are the individual parsed router configuration files, containing information such as PacketSets for each ACL, raw parsed data of each ACL, and interface information (stored by interface name, and interface ip).

4. Computing Reachability

After parsing is completed, the next stage of the automated solution is to perform reachability computations. The RUB of the system is computed, based on user selection of the source and destination routers. The output of the computation is stored in the earlier user-selected output directory. For each source-destination pair of routers selected, a correspondingly named output file would be created containing the results of the reachability bound computations.

5. Example Network

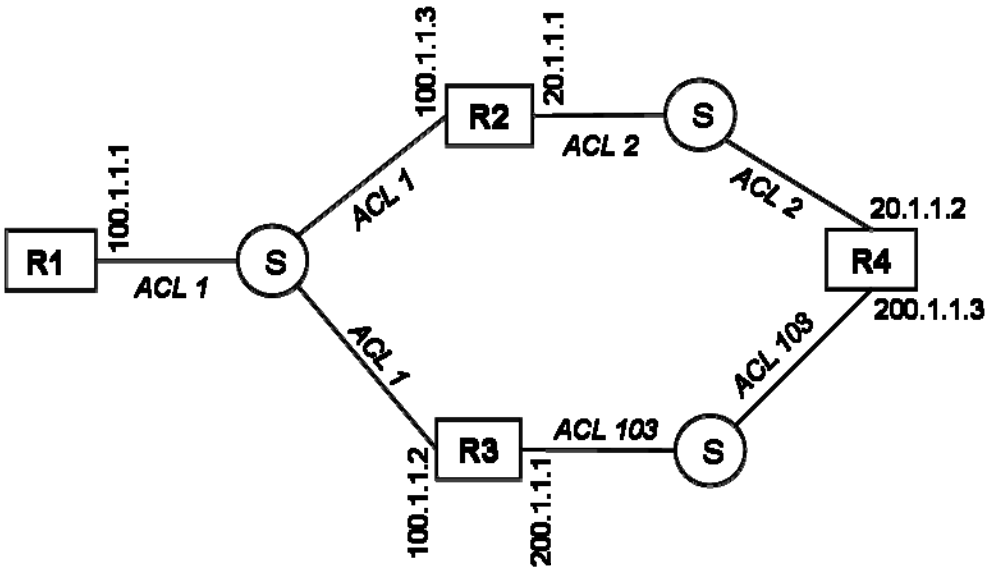

Figure 6. Example Network for Algorithm Illustration

Figure 6. shows an example network for the purpose of illustrating the algorithm for the computation of the reachability bound. There are 3 sub-networks in the example network, with routers connected in each sub-network via a switch. This reflects how the system views each sub-network using the neighbor concept. For example, R1 has an IP address of 100.1.1.1, R2 has an IP address of 100.1.1.3, and R3 has an IP address of 100.1.1.2. Each of these routers (R1 to R3) has an ACL (ACL 1) applied on its inbound interface.

The table below details the ACLs that will be applied on the inbound interfaces on the routers in the above example network.

access-list 1	deny 10.0.0.0 0.255.255.255
access-list 1	deny 127.0.0.0 0.255.255.255
access-list 1	permit any
access-list 2	deny 136.142.0.0 0.0.255.255
access-list 2	permit 124.0.0.0 0.255.255.255
access-list 103	deny udp any any eq snmp
access-list 103	permit ip any any

Table 2. ACLs Used in the Example Network

Following the format of a 5-tuple outlined in Chapter III, an example of the output of the system would be:

Reachability Upper Bound from R1 to R4:

[0.0.0.0 , 9.255.255.255] ; [0 , 65535] ; [0.0.0.0 , 255.255.255.255] ; [0 , 65535] ; [0 , 255]

[11.0.0.0 , 126.255.255.255] ; [0 , 65535] ; [0.0.0.0 , 255.255.255.255] ; [0 , 65535] ; [0 , 255]

[128.0.0.0 , 255.255.255.255] ; [0 , 65535] ; [0.0.0.0 , 255.255.255.255] ; [0 , 65535] ; [0 , 255]

6. Security Policy Implications

The above example showed how the Static Reachability Analysis Toolkit performs reachability computations based on the router configurations. However, it may not be apparent how security policies are affected.

The following example attempts to depict how changes in router configuration can affect security policy, how the security policy change can be overlooked, and how the Toolkit helps to detect the implication to the existing security policy.

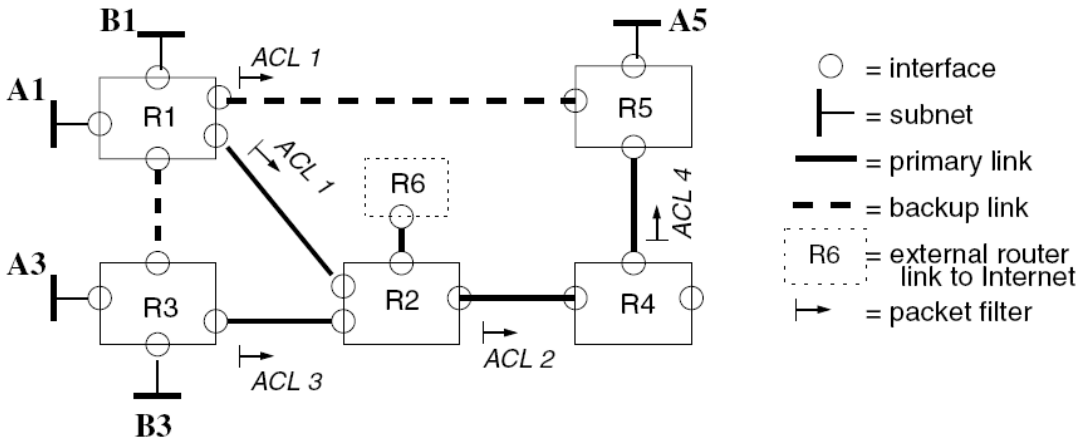

Figure 7. Example Enterprise Network (From [3])

This example is based on the one in [3], with an example enterprise network consisting of five routers, named R1 to R5. R1 and R3 are remote sales offices, and R2, R4, and R5 are at the central office. R6 is the gateway to the Internet. The network policy is to allow hosts on subnets A1 and A3 to communicate with the Microsoft SQL servers (port 1433) on subnet A5, and to prevent any other subnets from accessing the same servers. Also, policy dictates that all Sun ND protocol (protocol 77) packets be denied from entering the central office. In addition, multicast packets (in 224.0.0.0/8) are denied from leaving R3; and mobile IP (protocol 55) packets are also denied from reaching the servers. The referenced example also uses an implicit permit any rule as default.

The ACLs for the above are as follow:

```
ACL 1
access-list 101 permit tcp  110.0.0.0 0.255.255.255
     150.0.0.0 0.255.255.255 eq 1433
access-list 101 deny tcp any any eq 1433
access-list 101 permit ip any any

ACL 2
access-list 102 deny 77 any any
access-list 102 permit ip any any

ACL 3
access-list 103 permit tcp 130.0.0.0 0.255.255.255
     150.0.0.0 0.255.255.255 eq 1433
access-list 103 deny tcp any any eq 1433
access-list 103 deny ip any 224.0.0.0 0.255.255.255
access-list 103 permit ip any any

ACL 4
access-list 104 deny 55 any any
access-list 104 permit ip any any
```

Taking the example of one RUB, from R3 to R5, packets allowed would belong to:

```
[ 0.0.0.0 , 255.255.255.255 ] ; [ 0 , 65535 ] ; [ 0.0.0.0 ,
     223.255.255.255 ] ; [ 0 , 1432 ] ; [ 0 , 54 ]
```

```
[ 0.0.0.0 , 255.255.255.255 ] ; [ 0 , 65535 ] ; [ 0.0.0.0 ,
     223.255.255.255 ] ; [ 0 , 65535 ] ; [ 56 , 76 ]
[ 0.0.0.0 , 255.255.255.255 ] ; [ 0 , 65535 ] ; [ 0.0.0.0 ,
     223.255.255.255 ] ; [ 0 , 65535 ] ; [ 78 , 255 ]
[ 0.0.0.0 , 255.255.255.255 ] ; [ 0 , 65535 ] ; [ 0.0.0.0 ,
     223.255.255.255 ] ; [ 1433 , 1433 ] ; [ 0 , 5 ]
[ 0.0.0.0 , 255.255.255.255 ] ; [ 0 , 65535 ] ; [ 0.0.0.0 ,
     223.255.255.255 ] ; [ 1433 , 1433 ] ; [ 7 , 54 ]
[ 0.0.0.0 , 255.255.255.255 ] ; [ 0 , 65535 ] ; [ 0.0.0.0 ,
     223.255.255.255 ] ; [ 1434 , 65535 ] ; [ 0 , 54 ]
[ 0.0.0.0 , 255.255.255.255 ] ; [ 0 , 65535 ];[ 225.0.0.0 ,
     255.255.255.255 ] ; [ 0 , 1432 ] ; [ 0 , 54 ]
[ 0.0.0.0 , 255.255.255.255 ] ; [ 0 , 65535 ];[ 225.0.0.0 ,
     255.255.255.255 ] ; [ 0 , 65535 ] ; [ 56 , 76 ]
[ 0.0.0.0 , 255.255.255.255 ] ; [ 0 , 65535 ];[ 225.0.0.0 ,
     255.255.255.255 ] ; [ 0 , 65535 ] ; [ 78 , 255 ]
[ 0.0.0.0 , 255.255.255.255 ] ; [ 0 , 65535 ];[ 225.0.0.0 ,
     255.255.255.255 ] ; [ 1433 , 1433 ] ; [ 0 , 5 ]
[ 0.0.0.0 , 255.255.255.255 ] ; [ 0 , 65535 ];[ 225.0.0.0 ,
     255.255.255.255 ] ; [ 1433 , 1433 ] ; [ 7 , 54 ]
[ 0.0.0.0 , 255.255.255.255 ] ; [ 0 , 65535 ];[ 225.0.0.0 ,
     255.255.255.255 ] ; [ 1434 , 65535 ] ; [ 0 , 54 ]
[ 130.0.0.0 , 130.255.255.255 ] ; [ 0,65535 ];[ 150.0.0.0 ,
     150.255.255.255 ] ; [ 1433 , 1433 ] ; [ 6 , 6 ]
```

Now, take the case where the two backup links (dotted lines) were added. The network operators felt that the security policy would be satisfied if the routing parameters on the routers were reconfigured to use the backup links under failure scenarios only. It should work, since the security policy was previously satisfied.

However, after performing a reachability bound computation, the above design is shown to be flawed. Looking at the same RUB for R3 to R5, multicast traffic would leave R3, which is against the security policy (previously satisfied by ACL 3):

```
[ 0.0.0.0 , 255.255.255.255 ] ; [ 0 , 65535 ] ; [ 0.0.0.0 ,
     255.255.255.255 ] ; [ 0 , 1432 ] ; [ 0 , 255 ]
[ 0.0.0.0 , 255.255.255.255 ] ; [ 0 , 65535 ] ; [ 0.0.0.0 ,
     255.255.255.255 ] ; [ 1433 , 1433 ] ; [ 0 , 5 ]
[ 0.0.0.0 , 255.255.255.255 ] ; [ 0 , 65535 ] ; [ 0.0.0.0 ,
     255.255.255.255 ] ; [ 1433 , 1433 ] ; [ 7 , 255 ]
```

```
[ 0.0.0.0 , 255.255.255.255 ] ; [ 0 , 65535 ] ; [ 0.0.0.0 ,
     255.255.255.255 ] ; [ 1433 , 65535 ] ; [ 56 , 76 ]
[ 0.0.0.0 , 255.255.255.255 ] ; [ 0 , 65535 ] ; [ 0.0.0.0 ,
     255.255.255.255 ] ; [ 1433 , 65535 ] ; [ 78 , 255 ]
[ 0.0.0.0 , 255.255.255.255 ] ; [ 0 , 65535 ] ; [ 0.0.0.0 ,
     255.255.255.255 ] ; [ 1434 , 65535 ] ; [ 0 , 255 ]
[ 110.0.0.0 , 110.255.255.255 ] ; [ 0 , 65535];[ 150.0.0.0,
     150.255.255.255 ] ; [ 1433 , 1433 ] ; [ 6 , 6 ]
[ 130.0.0.0 , 130.255.255.255 ] ; [ 0 , 65535];[ 150.0.0.0,
     150.255.255.255 ] ; [ 1433 , 1433 ] ; [ 6 , 6 ]
```

In this scenario, tools (e.g., traceroute, ping) that experimentally test the reachability between two hosts would not be able to verify whether the backup link configuration satisfied the security policy, unless the main links were shut down.

Thus we can see how the use of this Toolkit can help network designers/operators detect problems with security policy requirements early.

7. Class Diagram

The class diagram of the software is shown below:

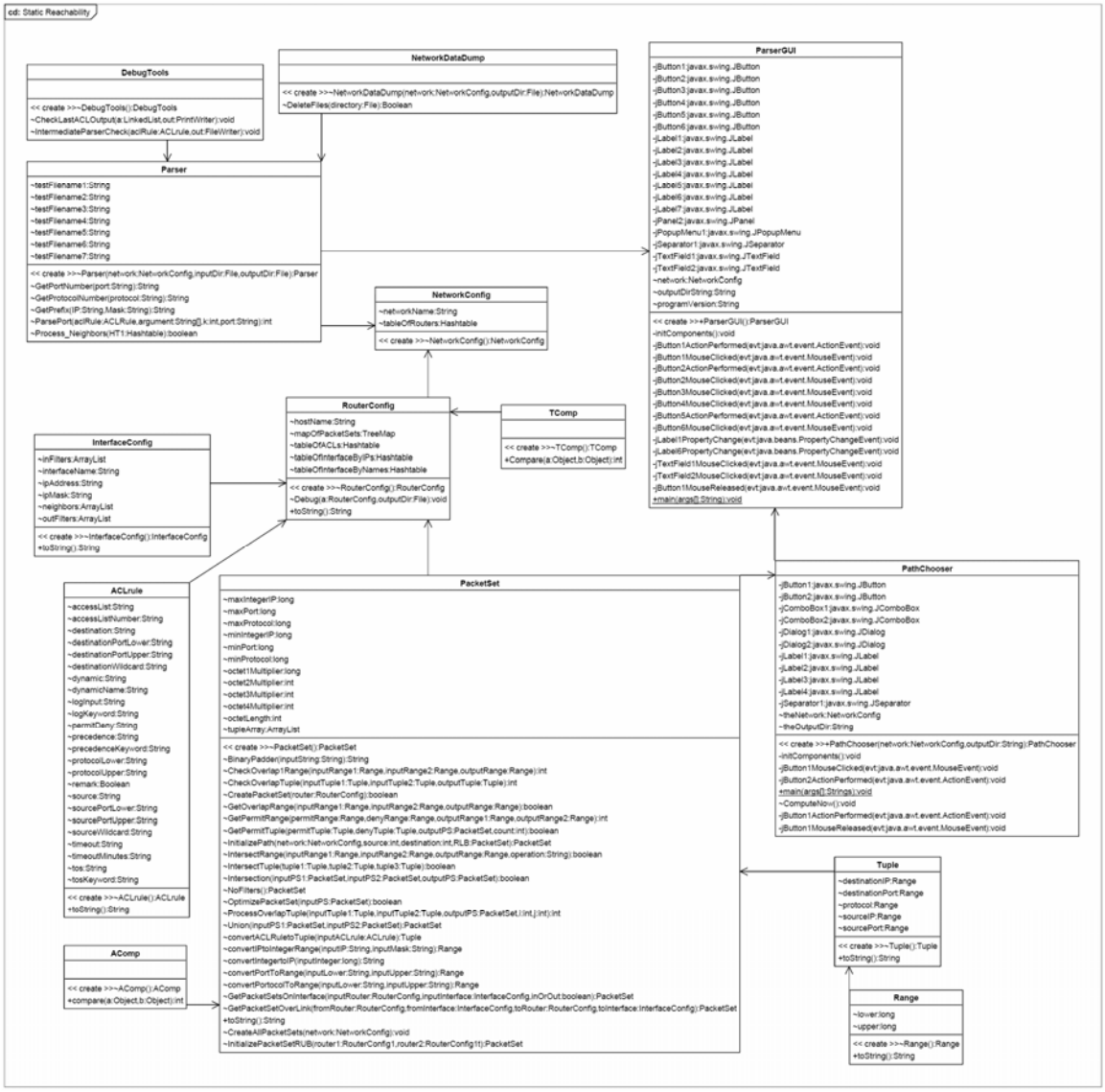

Figure 8. Class Diagram

8. Source Code

The code for the system software is organized by classes and is attached in Appendix C.

The first step of parsing the router configuration files stores all the parameters as a string representation in the various classes for easy future manipulation. The ACL rules were stored in a class which preserved the ordering of the rules, because they have to be processed sequentially.

All the ranges in the PacketSet representation are specified with a (*long*) *integer*. The purpose of using a *long* is to make it easy to manipulate IP addresses, as well as port and protocol numbers. IP addresses are converted to *long* by first converting them from their dotted decimal representation to their bit-string representation, and subsequently converting their bit-string (base two) to an integer (base ten). Port and protocol names are converted to their number equivalent using a lookup table [13, 14].

B. TRIAL WITH REAL WORLD ROUTER CONFIGURATIONS

Although it was not an objective to analyze a set of router configuration files, testing was done with real world router configuration files from Carnegie-Mellon University (CMU).

A set of twenty router configuration files from CMU was run on a Pentium M 1.86 GHz system with 1 GB of RAM, and the following was observed.

1. Processing Time

For the parsing stage, with the set of twenty router configuration files, it took less than one second to complete parsing these files, and creating the corresponding PacketSets for every ACL (packet filter).

For the reachability computation stage, it took less than three seconds to complete the computations for any source-destination pair.

2. PacketSet Efficiency

The set of twenty router configuration files from CMU had a total of 204 ACLs, and 1,332 ACL rules (rules in each packet filter). From this set, the Toolkit generated 2,245 PacketSet elements, amounting to 168.54% of the number of

ACL rules. However, the mean of the number of PacketSet elements to the number of ACL rules is much lower, at 127.53%. This means that the PacketSet representation, which already allows for a much more flexible representation of packet filter rules, is also a rather efficient representation in terms of storage.

It was also observed that there is no direct correspondence between the number of ACL rules to the number of PacketSet elements generated. For instance, an ACL with 15 ACL rules was represented by just 10 PacketSet elements. Yet on another occasion, an ACL with 14 ACL rules required 126 PacketSet elements for representation. The number of PacketSet elements depends more on the complexity of the ACL rules, i.e., interleaving of permit and deny rules and granularity of each rule. The more granular each rule is, and the more interleaved permit and deny rules are, the greater the number of PacketSet elements.

See Appendix D for more details of the data set.

C. CONCLUSION

This chapter provided an account of how the Toolkit should be used to compute RUBs. Two examples were given to illustrate the operation of the Toolkit with router configuration files from a network. The next chapter will sum up the entire thesis effort, and provide some recommendations for future work.

V. CONCLUSION AND RECOMMENDATIONS

This thesis has partially validated the static reachability analysis framework developed in [3] through a Java implementation. It demonstrates that the effects of statically configured packet filters on the reachability upper bounds of a network can be computed efficiently. This is an exciting step, because the implementation may be extended into a complete automated solution for static analysis of network reachability. The solution will be critical for verifying that the security policy of a network is satisfied by the network's design, under routine conditions, as well as failure scenarios.

The use of a PacketSet (defined in Chapter III) is crucial for performing operations such as union and intersection of packet filters across an entire network. Hence the creation of the PacketSet data structure was a key factor in the success of this research. By mapping all packet filter rules consisting of both permits and denies into a single PacketSet consisting of purely "permits", makes it an easy and flexible data structure to use.

This Toolkit has shown that it is also possible to compute finer-grain RUBs. For instance, whether a particular type of packet like port 80 traffic will be permitted between two hosts can be determined just by looking at the RUB results.

For future work, in addition to performing more rigorous tests of the Static Reachability Analysis Toolkit developed in this thesis with larger networks, the RLB algorithm outlined in [3] would have to be studied and validated for use with an automated solution.

At the time of writing, AT&T researchers were requested to assist in the testing of the developed Toolkit with larger networks, as well as collaborate on the verification of the routing protocol mechanism for use in an automated solution in conjunction with another thesis effort.

Therefore, work on the development of an automated solution that also considers the effects of routing protocols and packet transformations, to complete the automation of the static analysis framework [3], can now proceed. Thus, one future research area is the implementation of the techniques outlined in [3] with Java, to extend the features of the Static Reachability Analysis Toolkit.

Another possible research area is the development of a more efficient data structure to replace the tuple data structure used in this thesis, which may be beneficial when applied to very large networks, used in conjunction with an automated solution that considers packet filters, routing protocols and packet transformations.

With the development of the Static Reachability Analysis Toolkit, the state-of-the-art is one step closer towards having networks that completely satisfy their security policies. Incorporating routing protocol and packet transformation aspects in the Toolkit will enable networks of the future to be more secure, because human design error will be mitigated with this verification tool.

APPENDIX A. SYNTAX EXPLANATION

The explanation for the Cisco IOS syntax used in Chapter II is as follows (extracted from [15]):

Bold indicates commands and keywords. Example: **`access-list`**

Italics indicate user variables. Example : *`access-list-number`*

Braces ({ }) indicate a required choice. Example: {**`permit`** | **`deny`**}

Vertical bars (|) separate alternative, mutually exclusive arguments. Example: {**`permit`** | **`deny`**}

Square brackets ([ ]) indicate optional elements. Example: [**`tos`** *`tos`*]

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. SAMPLE ROUTER CONFIGURATION FILE

```
!
! Last configuration change at 12:00:00 PST Tue Nov 28 2006
! NVRAM config last updated at 12:00:00 PST Tue Nov 28 2006
!
version 12.2
no service pad
service timestamps debug datetime msec localtime show-
     timezone
service timestamps log datetime msec localtime show-
     timezone
service password DELETED
no service single-slot-reload-enable DELETED
!
hostname R2.gw
!
interface subnet1
 description R2 to R1
 ip address 10.0.0.2 255.255.255.252
 ip access-group 101 out
!
interface subnet4
 description R2 to R6
 ip address 40.0.0.1 255.255.255.252
!
interface subnet5
 description R2 to R4
 ip address 50.0.0.1 255.255.255.252
 ip access-group 102 out
!
interface subnet6
 description R2 to R3
 ip address 60.0.0.1 255.255.255.252
 ip access-group 103 out
!
access-list 101 permit tcp 110.0.0.0 0.255.255.255
     150.0.0.0 0.255.255.255 eq 1433
access-list 101 deny tcp any any eq 1433
access-list 101 permit ip any any
access-list 102 deny 77 any any
access-list 102 permit ip any any
access-list 103 permit tcp 130.0.0.0 0.255.255.255
     150.0.0.0 0.255.255.255 eq 1433
access-list 103 deny tcp any any eq 1433
access-list 103 deny ip any 224.0.0.0 0.255.255.255
```

```
access-list 103 permit ip any any
!
end
```

APPENDIX C. SOURCE CODES

A. ACLRULE.JAVA

```
package StaticReachabilityAnalysis;
/*
 * ACLrule.java
 *
 */

import java.util.*;

class ACLrule {
    String accessList;
    String accessListNumber;
    String dynamic;
    String dynamicName;
    String timeout;
    String timeoutMinutes;
    String permitDeny;
    String protocolLower;
    String protocolUpper;
    String source;
    String sourceWildcard;
    String sourcePortLower;
    String sourcePortUpper;
    String destination;
    String destinationWildcard;
    String destinationPortLower;
    String destinationPortUpper;
    String precedenceKeyword;
    String precedence;
    String tosKeyword;
    String tos;
    String logKeyword;
    String logInput;
    Boolean remark;

    ACLrule(){
        accessList = null;
        accessListNumber = null;
        dynamic = null;
        dynamicName = null;
        timeout = null;
        timeoutMinutes = null;
        permitDeny = null;
```

```
        protocolLower = null ;
        protocolUpper = null ;
        source = null;
        sourceWildcard = null;
        sourcePortLower = null;
        sourcePortUpper = null;
        destination = null;
        destinationWildcard = null;
        destinationPortLower = null ;
        destinationPortUpper = null ;
        precedenceKeyword = null;
        precedence = null;
        tosKeyword = null;
        tos = null;
        logKeyword = null;
        logInput = null;
        remark = false;
    }

    public String toString() {
        return accessList + " "
                + accessListNumber + " "
                + dynamic + " "
                + dynamicName + " "
                + timeout + " "
                + timeoutMinutes + " "
                + permitDeny + " "
                + protocolLower + " "
                + protocolUpper + " "
                + source + " "
                + sourceWildcard + " "
                + sourcePortLower + " "
                + sourcePortUpper + " "
                + destination + " "
                + destinationWildcard + " "
                + destinationPortLower + " "
                + destinationPortUpper + " "
                + precedenceKeyword + " "
                + precedence + " "
                + tosKeyword + " "
                + tos + " "
                + logKeyword + " "
                + logInput + " "
                + remark;
    }
} // end class ACLrule
```

B. ACOMP.JAVA

```
    package StaticReachabilityAnalysis;
/*
 * AComp.java
 *
 */

import java.util.*;

class AComp implements Comparator  {
    // This Comparator overrides the default compare function
    // Used for the ArrayList sorting in OptimizePacketSet
    // Enables correct sorting of the tuples
    public int compare (Object a, Object b) {
        int result=0;
        Tuple aTuple, bTuple;

        aTuple = (Tuple) a;
        bTuple = (Tuple) b;

        // if a is smaller than b, then a will come first
        // 1st Check : sourceIP.lower
        if (aTuple.sourceIP.lower == bTuple.sourceIP.lower ) {

          // If tied, do 2nd Check : souceIP.upper
          if (aTuple.sourceIP.upper == bTuple.sourceIP.upper) {

            // If tied again, do 3rd Check : sourcePort.lower
            if (aTuple.sourcePort.lower ==
                  bTuple.sourcePort.lower) {

              // If tied again, do 4th Check :sourcePort.upper
              if (aTuple.sourcePort.upper ==
                    bTuple.sourcePort.upper) {

                // If tied again, do 5th Check : distinationIP.lower
                if (aTuple.destinationIP.lower ==
                      bTuple.destinationIP.lower){

                  // If tied again, do 6th Check : distinationIP.upper
                  if (aTuple.destinationIP.upper ==
                        bTuple.destinationIP.upper){

                    // If tied again, do 7th Check : distinationPort.lower
                    if (aTuple.destinationPort.lower ==
```

```
            bTuple.destinationPort.lower){

          // If tied again, do 8th Check : distinationPort.upper
          if (aTuple.destinationPort.upper ==
                bTuple.destinationPort.upper){

            // If tied again, do 9th Check : protocol.upper
            if (aTuple.protocol.lower ==
                  bTuple.protocol.lower) {

              // If tied again, do 10th Check : protocol.upper
              if (aTuple.protocol.upper ==
                    bTuple.protocol.upper) result = 0;
              else if (aTuple.protocol.upper <
                    bTuple.protocol.upper) result = -1;
              else result = 1;

            }
            else if (aTuple.protocol.lower <
                  bTuple.protocol.lower) result = -1;
            else result = 1;
          }
          else if (aTuple.destinationPort.upper <
                bTuple.destinationPort.upper) result = -1;
          else result = 1;

        }
        else if (aTuple.destinationPort.lower <
              bTuple.destinationPort.lower) result = -1;
        else result = 1 ;
      }
      else if (aTuple.destinationIP.upper <
            bTuple.destinationIP.upper) result = -1;
      else result = 1 ;
    }
    else if (aTuple.destinationIP.lower <
          bTuple.destinationIP.lower) result = -1;
    else result = 1;
  } // end of 3rd Check
  else if (aTuple.sourcePort.upper <
        bTuple.sourcePort.upper) result = -1;
  else result = 1 ;

} // end of 2nd Check
else if (aTuple.sourcePort.lower <
       bTuple.sourcePort.lower)  result = -1;
```

```
            else result = 1 ;
          }
          else if (aTuple.sourceIP.upper <
                bTuple.sourceIP.upper) result = -1 ;
          else result = 1;
        }
        else if (aTuple.sourceIP.lower <
              bTuple.sourceIP.lower) result = -1 ;
        else result = 1;

        return result;
    }
    /** Creates a new instance of AComp */
    AComp() {
    }
}
```

C. DEBUGTOOLS.JAVA

```
package StaticReachabilityAnalysis;
/*
 * DebugTools.java
 *
 */

import java.util.*;
import java.io.*;

class DebugTools {

    /** Creates a new instance of DebugTools */
    DebugTools() {
    }

    /*****************************************************
     *
     * Writes the last ACL parsed into an output file
     *
     *****************************************************/
    void CheckLastACLOutput(LinkedList a, PrintWriter out) {
        try {
            out.println("Parsed Output");
            out.println("access-list|" + " # |" + "dyn & #|"
                    + "timeout & #|" + "permorden|"
                    + " prot|" + " source & wc|"
                    + " dest & wc|" + " prcd & #|" + " tos & #|"
                    + " log & input ");
            Iterator itr = a.iterator();
            while (itr.hasNext()) {
               out.println (itr.next());
            }
        } catch (Exception e) { System.out.println ("Error - " + e); }
    }

    /*****************************************************
     *
     * Writes the ACL rule parsed into an output file
     *
     *****************************************************/
    void IntermediateParserCheck(ACLrule aclRule, FileWriter out){
        try {
            out.write (aclRule.accessList + " | ");
            out.write (aclRule.accessListNumber + " | ");
            out.write (aclRule.permitDeny + " | ");
```

```
            out.write (aclRule.protocolLower + " | ");
            out.write (aclRule.source + " | ");
            out.write (aclRule.sourceWildcard + " | ");
            out.write (aclRule.destination + " | ");
            out.write (aclRule.destinationWildcard + " | ");
            out.write ("\r\n" );
        } catch (Exception e) { System.out.println ("Error - " + e); }
    }
}
```

D. INTERFACECONFIG.JAVA

```
package StaticReachabilityAnalysis;
/*
 * InterfaceConfig.java
 *
 */

import java.util.*;

class InterfaceConfig {
    String interfaceName; // Interface name
    String ipAddress; // IP address - yet to handle secondary case!
    String ipMask;  // IP mask
    ArrayList neighbors; // Array of neighbors --> make this array of objects
                // First check whether IP mask is the same length or same?
                // If same, check whether same network prefix (ip address + ip mask)
                // If not same --> not neighbors
                // If same network prefix --> neighbors
    ArrayList inFilters; // Array of incoming filters
    ArrayList outFilters; // Array of outgoing filters

    /** Creates a new instance of InterfaceConfig */
    InterfaceConfig() {
        interfaceName = null;
        ipAddress = null;
        ipMask = null; // Process_Neighbors
        //   [if   (currentInterface.ipMask.equals(interfaceToCompare.ipMask))]   does
not work if set to null
        neighbors = new ArrayList();
        inFilters = new ArrayList();
        outFilters = new ArrayList();
    }

    public String toString() {
        return interfaceName + " " + ipAddress + " " + ipMask + " " + neighbors + " "
                + inFilters + " " + outFilters;
    }
}
```

E. NETWORKCONFIG.JAVA

```
package StaticReachabilityAnalysis;
/*
 * NetworkConfig.java
 *
 */

import java.util.*;
import java.io.*;

class NetworkConfig {

    String networkName; // Network name
    Hashtable tableOfRouters; // Table of pointers to Routers

    /** Creates a new instance of NetworkConfig */
    NetworkConfig() {
        networkName = null;
        tableOfRouters = new Hashtable();
    }
}
```

F. NETWORKDATADUMP.JAVA

```
package StaticReachabilityAnalysis;
/*
 * NetworkDataDump.java
 *
*/

import java.util.*;
import java.io.*;

class NetworkDataDump {

   /** Creates a new instance of NetworkDataDump */
   NetworkDataDump(NetworkConfig network, File outputDir) {
      try {
         DeleteFiles (outputDir);
         String outputFileName = "Network Data.txt";
         File outputFile = new File (outputDir, outputFileName);
         FileWriter outFile = new FileWriter (outputFile);

         // Data to be written to file
         outFile.write("Network ID : " + network.networkName + "\r\n\r\n");

         // test output to screen
         System.out.println("Network ID : " + network.networkName);

         // Write Router Data
         outFile.write("Router List : \r\n");

         System.out.println("Router List :  "); // test output to screen

         int routerCounter=0;
         RouterConfig currentRouter;
         Enumeration routerList = network.tableOfRouters.elements();

         while ( routerList.hasMoreElements() ) {
            currentRouter = (RouterConfig) routerList.nextElement();
            routerCounter++;
            outFile.write("(" + routerCounter + ") " +
                  currentRouter.hostName + "\r\n");

            System.out.println("(" + routerCounter + ") " +
                  currentRouter.hostName); // test output to screen

            currentRouter.Debug(currentRouter, outputDir);
         }
```

```
            // Close the file written to
            outFile.close();
        } catch (Exception e) { System.out.println ("Error - " + e); }
    }

    boolean DeleteFiles(File directory) {
        boolean success = false;
        if (directory.exists()) {
            File[] files = directory.listFiles();
            for (int i=0; i<files.length; i++) {
                if (files[i].isDirectory()) {
                    success = DeleteFiles(files[i]);
                }
                else {
                    files[i].delete();
                }
            }
        }
        return success;
    }
}
```

G. PACKETSET.JAVA

```
package StaticReachabilityAnalysis;
/*
 * PacketSet.java
 *
*/

import java.util.*;
import java.io.*;
import java.math.*;

class PacketSet {

    ArrayList tupleArray ;
    int octetLength = 8 ;
    long octet1Multiplier = 16777216L ;
    int octet2Multiplier = 65536 ;
    int octet3Multiplier = 256 ;
    int octet4Multiplier = 1 ;
    long minIntegerIP = 0L ;
    /* 4294967295 is max value of an IP (255.255.255.255) in decimal value */
    long maxIntegerIP = 4294967295L ;
    long minPort = 0L;
    long maxPort = 65535L;
    long minProtocol = 0;
    long maxProtocol = 255;

    /*** Creates a new instance of PacketSet ***/
    PacketSet() {
        tupleArray = new ArrayList();
    }

    /*** Set what to return when a function wants to display a PacketSet ***/
    public String toString (){
        String output= new String();
        for (int counter=0; counter<tupleArray.size();counter++)
            output = output + tupleArray.get(counter) + " \r\n ";
        return output;
    }


    /**************************************************************************
     *
     * Create PacketSets for every router in the network object
     * Determine how many routers are in the network object
     * For every router, run a CreatePacketSet function on it
```

```
 *
 ***************************************************************************/
void CreateAllPacketSets (NetworkConfig network) {
   /*****************************************
    * Display for user
    *****************************************/
   System.out.println("\r\n---------------------------------------" +
         "----------------------------\r\n");
   System.out.println("======= Creating Packet Sets =======");
   /*****************************************
    * Get the list of routers
    * Create PacketSets for each router
    *****************************************/
   Enumeration routerList = network.tableOfRouters.elements();
   while ( routerList.hasMoreElements() ) {
      boolean success = CreatePacketSet
            ((RouterConfig) routerList.nextElement());
      if (success) System.out.println
            ("       - Packet Set created successfully ");
      else System.out.println ("       - No Packet Set created");
   }
   /*****************************************
    * Display for user
    *****************************************/
   System.out.println("\r\n---------------------------------------" +
         "--------------------------\r\n");
}

/***************************************************************************
 *
 * Generate PacketSets for a RouterConfig object.
 * Each PacketSet corresponds to an ACL stored in the RouterConfig object.
 * This object generated is stored in the same RouterConfig object.
 * A deny buffer is temporarily created to hold any deny rules.
 * Store the array of tuples for each ACL in a permit buffer.
 * Store final permit buffer in RouterConfig object
 *      (as a TreeMap) with ACL number as the key.
 *
***************************************************************************/
boolean CreatePacketSet (RouterConfig router) {
   boolean success = false;
   PacketSet permitBuffer, denyBuffer, interimBuffer  ;
   String acl = new String();
   Tuple currentTuple;
   LinkedList aclList;
   ACLrule aclRule;
```

```
Enumeration aclNumber = router.tableOfACLs.keys();
System.out.println("<<< Router : " + router.hostName + " >>>");

/**********************************
 * Process all the elements in the ACL table, in order of the keys
 * Only one element is processed at a time.
 **********************************/
while(aclNumber.hasMoreElements()) {
   /*** Get the next ACL ***/
   acl = (String) aclNumber.nextElement();// Get the next ACL

   /*** Get the linked list of each ACL ***/
   aclList = (LinkedList) router.tableOfACLs.get(acl);

   /*** Create a new deny and permit buffer ***/
   denyBuffer = new PacketSet();
   permitBuffer = new PacketSet();

   /*** Process the ACL rules in the ACL, one at a time ***/
   for (int counter=0; counter < aclList.size(); counter++) {
      /*** Initialization ***/
      currentTuple = new Tuple();
      interimBuffer = new PacketSet();
      aclRule = (ACLrule) aclList.get(counter);

      /**************************************
       * Process current ACL rule into a tuple
       * Extract the 5 ranges from the ACL rule
       *      and store in currentTuple
       **************************************/
      currentTuple.sourceIP = convertIPtoIntegerRange
            (aclRule.source, aclRule.sourceWildcard);
      currentTuple.sourcePort = convertPortToRange
            (aclRule.sourcePortLower, aclRule.sourcePortUpper);
      currentTuple.destinationIP = convertIPtoIntegerRange
            (aclRule.destination, aclRule.destinationWildcard);
      currentTuple.destinationPort = convertPortToRange
            (aclRule.destinationPortLower, aclRule.destinationPortUpper);
      currentTuple.protocol = convertProtocolToRange
            (aclRule.protocolLower, aclRule.protocolUpper);


      /*** If ACL rule is a deny, add the
       * current tuple to the deny buffer ***/
      if (aclRule.permitDeny.equalsIgnoreCase("deny"))
         denyBuffer.tupleArray.add(currentTuple);
```

```
    /*** Otherwise if ACL rule is a permit, do the following ***/
    else if (aclRule.permitDeny.equalsIgnoreCase("permit")) {
       interimBuffer.tupleArray.add(currentTuple);

       /*******************************
        * If there are no deny rules,
        * add currentTuple to denyBuffer straightaway
        *******************************/
       if (denyBuffer.tupleArray.size()==0) {
          //permitBuffer.tupleArray.add(currentTuple);
          success = true;
       }

       /*******************************
        * Compare current permit tuple against each tuple
        * in the deny buffer one at a time
        *******************************/
       for (int i=0; i<denyBuffer.tupleArray.size(); i++) {
          Tuple outputTuple = new Tuple();
          boolean toDecrement = false;
          /*** Get the next tuple in the deny buffer ***/
          Tuple denyTuple = (Tuple) denyBuffer.tupleArray.get(i);

          for (int j=0; j<interimBuffer.tupleArray.size(); j++) {
             toDecrement = interimBuffer.GetPermitTuple(
                   (Tuple)interimBuffer.tupleArray.get(j),
                   denyTuple,interimBuffer,j);
             success = true;
             /*** if object at position j was changed,
              * decrement j by 1
              * so as to process the new object ***/
             if (toDecrement) j-- ;
          }
       }
    } // end of else if (aclRule.permitDeny.equalsIgnoreCase("permit"))

    /*** Add the interim buffer contents to the permit buffer ***/
    permitBuffer.tupleArray.addAll(interimBuffer.tupleArray);
} // end of for (int counter=0; counter < aclList.size(); counter++)

/*****************************************
 * If there are ACLs in the Permit Buffer,
 * - Run the optimizer
 * - Add the permit buffer to the router object
 *****************************************/
```

```
        if (permitBuffer.tupleArray.size()>0) {
            // Optimize the Permit Buffer
            boolean isSuccess=false;
            isSuccess = OptimizePacketSet(permitBuffer);

            /****************************************
             * Store the permitBuffer in the
             * routerConfig object Map, using acl as key
             ****************************************/
            router.mapOfPacketSets.put(acl, permitBuffer);
        }
    } // end of while(aclNumber.hasMoreElements())

    return success;
} // end of CreatePacketSet


 /*************************************************************************
 *
 * Optimize a PacketSet
 * - remove empty tuples
 * - remove completely overlapping tuples
 * - merge adjacent tuples
 * - sort the tuples
 *
*************************************************************************/
boolean OptimizePacketSet (PacketSet inputPS) {
    boolean success = false;

    /**************************
     * Remove any empty tuples in the PacketSet
     **************************/
    for (int i=0; i<inputPS.tupleArray.size(); i++) {
        Tuple currentTuple = (Tuple) inputPS.tupleArray.get(i);
        if (currentTuple.sourceIP.lower == -1) {
            inputPS.tupleArray.remove(i);
            i--;
            success = true;
        }
    }

    /**************************
     * Loop through looking for overlapping tuples
     **************************/
    boolean toContinue;
    do {
```

```
        toContinue = false;

        /*** Use a double loop to process the elements in the PacketSet ***/
        for (int i=0; i<inputPS.tupleArray.size()-1; i++) {

            for (int j=i+1; j<inputPS.tupleArray.size(); j++) {
                Tuple tuple1 = (Tuple) inputPS.tupleArray.get(i) ;
                Tuple tuple2 = (Tuple) inputPS.tupleArray.get(j) ;
                int decJ = ProcessOverlapTuple(tuple1,tuple2,inputPS,i,j);
                if (decJ>=1) { // if there was a removal of any tuple
                    toContinue = true;
                    if (decJ==2) j--;
                }
            } // end of inner for loop
            if (toContinue) success = true;
        } // end of outer for loop
    } while (toContinue);


    /******************
     * Sort the tuples
     ******************/
    Object interimArray[] = inputPS.tupleArray.toArray();
    Arrays.sort(interimArray, new AComp());
    for (int i=0; i<interimArray.length; i++)
        inputPS.tupleArray.set(i,interimArray[i]);
    return success;
}


/**************************************************************************
 *
 * Check 2 tuples for overlaps
 * Remove a tuple that is completely overlapped by another
 * Merge adjacent tuples
 * Do nothing about partially overlapping tuples
 *      (takes more storage otherwise)
 *
 **************************************************************************/
int ProcessOverlapTuple (Tuple inputTuple1, Tuple inputTuple2,
    PacketSet outputPS, int i, int j) {
    Tuple outputTuple = new Tuple();
    /*** decJ is used to signal whether there was any successful removal
     * or merging of tuples ***/
    int decJ = 0;
```

```
/*************************************************************
 * Check whether inputTuple1 and inputTuple2 have overlaps
 * store new tuple in outputTuple
 * return value is result
 * Process tuples according to the return value (result)
 *************************************************************/
int result = CheckOverlapTuple (inputTuple1, inputTuple2, outputTuple);

switch (result) {
   case 0 : break; // do nothing;
   /*********************************************************
   * tuple1 completely covers tuple2. outputTuple can be used
   *********************************************************/
   case 243 :
   case 405 :
   case 675 :
   case 1125 :
   case 1875 :

      outputPS.tupleArray.remove(j); // remove tuple2
      decJ=2;
      break;
   /*********************************************************
   * tuple2 completely covers tuple1. outputTuple can be used
   *********************************************************/
   case 1024 :
   case 1280 :
   case 1600 :
   case 2000 :
   case 2500 :
   case 3125:
      // remove tuple1
      outputPS.tupleArray.remove(i);
      // put tuple2 in tuple1's position
      outputPS.tupleArray.add(i,inputTuple2);
      // remove tuple2 from it's orginal position
      outputPS.tupleArray.remove(j);
      decJ=2;
      break;
   /*********************************************************
   * 1 of the ranges in both tuples are adjacent and can be combined
   * and the other ranges are exactly the same. outputTuple can be used
   *********************************************************/
   case 1250 :
      outputPS.tupleArray.add(i,outputTuple);
      outputPS.tupleArray.remove(i+1);
```

```
            outputPS.tupleArray.remove(j);
            decJ=1;
            break;
        /*********************************************************
        * at least 1 range in both tuples overlap partially
        *********************************************************/
        default : decJ=0; // do nothing
    }
    return decJ ;
}


 /*************************************************************************
 *
 * Function that checks all 5 ranges in each of the 2 tuples
 * Return the result of the check
 * As well as return the new tuple to outputTuple
 *
 *************************************************************************/
int CheckOverlapTuple (Tuple inputTuple1,
        Tuple inputTuple2, Tuple outputTuple) {
    int result1 = CheckOverlap1Range(inputTuple1.sourceIP,
            inputTuple2.sourceIP, outputTuple.sourceIP) ;
    int result2 = CheckOverlap1Range(inputTuple1.sourcePort,
            inputTuple2.sourcePort, outputTuple.sourcePort) ;
    int result3 = CheckOverlap1Range(inputTuple1.destinationIP,
            inputTuple2.destinationIP, outputTuple.destinationIP) ;
    int result4 = CheckOverlap1Range(inputTuple1.destinationPort,
            inputTuple2.destinationPort, outputTuple.destinationPort) ;
    int result5 = CheckOverlap1Range(inputTuple1.protocol,
            inputTuple2.protocol, outputTuple.protocol);
    int output = 0;
    /*********************************************
     * if output=1024, 1280, 1600, 2000, or 2500, and 3125,
     *      tuple2 completely covers tuple1.
     *      outputTuple can be used
     * if output=243, 405, 675, 1125, or 1875,
     *      tuple1 completely covers tuple2.
     *      outputTuple can be used
     * if output=1250, 1 of the ranges in both tuples are
     *      adjacent and can be combined
     *      and the other ranges are exactly the same.
     *      outputTuple can be used
     * if output=1, all ranges in both tuples overlap partially.
     *      Need to use another function
     * if output=0, at least 1 range does not overlap.
```

```
   ***********************************************/

   /*** Compact the return value ***/
   output = result1 * result2 * result3 * result4 * result5;

   return output;
}

 /*************************************************************************
 *
 * Function that compares 2 ranges for an overlap
 * Return the result of the check
 * As well as return the new range to outputRange
 *
 *************************************************************************/
int CheckOverlap1Range (Range inputRange1, Range inputRange2,
       Range outputRange) {
   long lower=0L, upper=0L;
   int output=0;
   /*****************************************************
    * if output=5, range1 and range 2 are exactly the same
    * if output=4, range2 completely covers range1
    * if output=3, range1 completely covers range2
    * if output=2, ranges don't overlap, but they are
    *      adjacent and can be combined
    * if output=1, ranges overlap partially
    * if output=0, ranges do not overlap
    *****************************************************/

   /*****************************************************
    * Ranges do not overlap  i.e., result in output=0
    *****************************************************/
   if (inputRange1.upper + 1 < inputRange2.lower ||
         inputRange2.upper + 1 < inputRange1.lower ) {
      output=0;
   }
   /*****************************************************
    * Both ranges are exactly the same
    *****************************************************/
   else if (inputRange1.lower == inputRange2.lower &&
         inputRange1.upper == inputRange2.upper) {
      outputRange.lower = inputRange1.lower;
      outputRange.upper = inputRange1.upper;
      output=5 ;
   }
   /*****************************************************
```

```
 * For all other combinations
 ************************************************/
else {
   /*********************************************
    * Determine which is the lower value of both ranges
    *********************************************/
   if (inputRange1.lower < inputRange2.lower) {
      outputRange.lower = inputRange1.lower;
      lower = inputRange2.lower;
   }
   else {
      outputRange.lower = inputRange2.lower;
      lower = inputRange1.lower;
   }
   /*********************************************
    * Determine which is the upper value of both ranges
    *********************************************/
   if (inputRange1.upper < inputRange2.upper) {
      outputRange.upper = inputRange2.upper;
      upper = inputRange1.upper;
   }
   else {
      outputRange.upper = inputRange1.upper;
      upper = inputRange2.upper;
      /* // For debugging
       System.out.println("output upper : " + outputRange.upper
       + " ; upper : " + upper);
       */
   }

   /*********************************************
    * Check if upper value is >= lower value
    * If so, it is a valid range
    * Otherwise, signal the calling function to discard
    *      the output range
    *********************************************/
   if (upper >= lower) {
      if (inputRange2.lower == lower &&
            inputRange2.upper==upper) output=3 ;
      else if (inputRange1.lower == lower &&
            inputRange1.upper==upper) output=4 ;
      else output=1;
   }
   /*********************************************
    * Check if both ranges are adjacent
    * Signal accordingly
```

```
        ***********************************************/
       else if (upper + 1 == lower) output=2;
    } // end else
    return output;
}


 /*************************************************************************
 *
 * Function that compares a permit tuple against a deny tuple
 * Saves the resultant permitted PacketSet to outputPS
 *
 * If at least 1 dimension/range doesn't overlap then nothing will be
 *     added to outputPS
 * outputPS doesn't need a value, because calling function uses a value
 *     in outputPS as permitTuple
 * count is the position of the permitTuple in outputPS
 *
 *************************************************************************/
boolean GetPermitTuple (Tuple permitTuple, Tuple denyTuple,
        PacketSet outputPS, int count) {
    boolean decrementCounter = false;
    Range outRange1 = new Range();
    Range outRange2 = new Range();
    Range outRange3 = new Range();
    Range outRange4 = new Range();
    Range outRange5 = new Range();
    Range outRange6 = new Range();
    Range outRange7 = new Range();
    Range outRange8 = new Range();
    Range outRange9 = new Range();
    Range outRange10 = new Range();
    int result1=0, result2=0, result3=0, result4=0, result5=0;
    Tuple tuple1 = new Tuple();
    Tuple tuple2 = new Tuple();
    Tuple tuple3 = new Tuple();
    Tuple tuple4 = new Tuple();
    Tuple tuple5 = new Tuple();
    Tuple tuple11 = new Tuple();
    Tuple tuple22 = new Tuple();
    Tuple tuple33 = new Tuple();
    Tuple tuple44 = new Tuple();
    Tuple tuple55 = new Tuple();

    result1 = GetPermitRange (permitTuple.sourceIP,
          denyTuple.sourceIP, outRange1, outRange2);
```

```
result2 = GetPermitRange (permitTuple.destinationIP,
        denyTuple.destinationIP, outRange3, outRange4);
result3 = GetPermitRange (permitTuple.sourcePort,
        denyTuple.sourcePort, outRange5, outRange6);
result4 = GetPermitRange (permitTuple.destinationPort,
        denyTuple.destinationPort, outRange7, outRange8);
result5 = GetPermitRange (permitTuple.protocol,
        denyTuple.protocol, outRange9, outRange10);
if (result1>=0 && result2>=0 && result3>=0 &&
        result4>=0 && result5>=0) {
    Range temp = new Range();
    if (result1>=1) {
        tuple1.sourceIP = outRange1 ;
        tuple1.destinationIP = permitTuple.destinationIP;
        tuple1.sourcePort = permitTuple.sourcePort ;
        tuple1.destinationPort = permitTuple.destinationPort ;
        tuple1.protocol = permitTuple.protocol ;
        outputPS.tupleArray.add(tuple1);
    }
    if (result1==2) {
        tuple11.sourceIP = outRange2 ;
        tuple11.destinationIP = permitTuple.destinationIP;
        tuple11.sourcePort = permitTuple.sourcePort ;
        tuple11.destinationPort = permitTuple.destinationPort ;
        tuple11.protocol = permitTuple.protocol ;
        outputPS.tupleArray.add(tuple11);
    }
    if (result2>=1 ) {
        if (GetOverlapRange (permitTuple.sourceIP,
                denyTuple.sourceIP, temp)) {
            tuple2.sourceIP.lower = temp.lower ;
            tuple2.sourceIP.upper = temp.upper ;
        }
        tuple2.destinationIP = outRange3;
        tuple2.sourcePort = permitTuple.sourcePort ;
        tuple2.destinationPort = permitTuple.destinationPort ;
        tuple2.protocol = permitTuple.protocol ;
        outputPS.tupleArray.add(tuple2);
    }
    if (result2==2) {
        if (GetOverlapRange (permitTuple.sourceIP,
                denyTuple.sourceIP, temp)){
            tuple22.sourceIP.lower = temp.lower ;
            tuple22.sourceIP.upper = temp.upper ;
        }
        tuple22.destinationIP = outRange4;
```

```
    tuple22.sourcePort = permitTuple.sourcePort ;
    tuple22.destinationPort = permitTuple.destinationPort ;
    tuple22.protocol = permitTuple.protocol ;
    outputPS.tupleArray.add(tuple22);
}
if (result3>=1) {
    if (GetOverlapRange (permitTuple.sourceIP,
            denyTuple.sourceIP, temp)){
        tuple3.sourceIP.lower = temp.lower ;
        tuple3.sourceIP.upper = temp.upper ;
    }
    if (GetOverlapRange (permitTuple.destinationIP,
            denyTuple.destinationIP, temp)){
        tuple3.destinationIP.lower = temp.lower ;
        tuple3.destinationIP.upper = temp.upper ;
    }
    tuple3.sourcePort = outRange5 ;
    tuple3.destinationPort = permitTuple.destinationPort ;
    tuple3.protocol = permitTuple.protocol ;
    outputPS.tupleArray.add(tuple3);
}
if (result3==2) {
    if (GetOverlapRange (permitTuple.sourceIP,
            denyTuple.sourceIP, temp)){
        tuple33.sourceIP.lower = temp.lower ;
        tuple33.sourceIP.upper = temp.upper ;
    }
    if (GetOverlapRange (permitTuple.destinationIP,
            denyTuple.destinationIP, temp)){
        tuple33.destinationIP.lower = temp.lower ;
        tuple33.destinationIP.upper = temp.upper ;
    }
    tuple33.sourcePort = outRange6 ;
    tuple33.destinationPort = permitTuple.destinationPort ;
    tuple33.protocol = permitTuple.protocol ;
    outputPS.tupleArray.add(tuple33);
}
if (result4>=1) {
    if (GetOverlapRange (permitTuple.sourceIP,
            denyTuple.sourceIP, temp)){
        tuple4.sourceIP.lower = temp.lower ;
        tuple4.sourceIP.upper = temp.upper ;
    }
    if (GetOverlapRange (permitTuple.destinationIP,
            denyTuple.destinationIP, temp)){
        tuple4.destinationIP.lower = temp.lower ;
```

```
            tuple4.destinationIP.upper = temp.upper ;
        }
        if (GetOverlapRange (permitTuple.sourcePort,
                denyTuple.sourcePort, temp)){
            tuple4.sourcePort.lower = temp.lower ;
            tuple4.sourcePort.upper = temp.upper ;
        }
        tuple4.destinationPort = outRange7 ;
        tuple4.protocol = permitTuple.protocol ;
        outputPS.tupleArray.add(tuple4);
    }
    if (result4==2) {
        if (GetOverlapRange (permitTuple.sourceIP,
                denyTuple.sourceIP, temp)){
            tuple44.sourceIP.lower = temp.lower ;
            tuple44.sourceIP.upper = temp.upper ;
        }
        if (GetOverlapRange (permitTuple.destinationIP,
                denyTuple.destinationIP, temp)){
            tuple44.destinationIP.lower = temp.lower ;
            tuple44.destinationIP.upper = temp.upper ;
        }
        if (GetOverlapRange (permitTuple.sourcePort,
                denyTuple.sourcePort, temp)){
            tuple44.sourcePort.lower = temp.lower ;
            tuple44.sourcePort.upper = temp.upper ;
        }
        tuple44.destinationPort = outRange8 ;
        tuple44.protocol = permitTuple.protocol ;
        outputPS.tupleArray.add(tuple44);
    }
    if (result5>=1) {
        if (GetOverlapRange (permitTuple.sourceIP,
                denyTuple.sourceIP, temp)){
            tuple5.sourceIP.lower = temp.lower ;
            tuple5.sourceIP.upper = temp.upper ;
        }
        if (GetOverlapRange (permitTuple.destinationIP,
                denyTuple.destinationIP, temp)) {
            tuple5.destinationIP.lower = temp.lower ;
            tuple5.destinationIP.upper = temp.upper ;
        }
        if (GetOverlapRange (permitTuple.sourcePort,
                denyTuple.sourcePort, temp)){
            tuple5.sourcePort.lower = temp.lower ;
            tuple5.sourcePort.upper = temp.upper ;
```

```
            }
            if (GetOverlapRange (permitTuple.destinationPort,
                  denyTuple.destinationPort, temp)){
               tuple5.destinationPort.lower = temp.lower ;
               tuple5.destinationPort.upper = temp.upper ;
            }
            tuple5.protocol = outRange9 ;
            outputPS.tupleArray.add(tuple5);
         }
         if (result5==2) {
            if (GetOverlapRange (permitTuple.sourceIP,
                  denyTuple.sourceIP, temp)){
               tuple55.sourceIP.lower = temp.lower ;
               tuple55.sourceIP.upper = temp.upper ;
            }
            if (GetOverlapRange (permitTuple.destinationIP,
                  denyTuple.destinationIP, temp)){
               tuple55.destinationIP.lower = temp.lower ;
               tuple55.destinationIP.upper = temp.upper ;
            }
            if (GetOverlapRange (permitTuple.sourcePort,
                  denyTuple.sourcePort, temp)){
               tuple55.sourcePort.lower = temp.lower ;
               tuple55.sourcePort.upper = temp.upper ;
            }
            if (GetOverlapRange (permitTuple.destinationPort,
                  denyTuple.destinationPort, temp)){
               tuple55.destinationPort.lower = temp.lower ;
               tuple55.destinationPort.upper = temp.upper ;
            }
            tuple55.protocol = outRange10 ;
            outputPS.tupleArray.add(tuple55);
         }
         outputPS.tupleArray.remove(count);
         decrementCounter = true ;
      } // end if

      return decrementCounter;
   } // end of GetPermitTuple

   /**********************************************************************
   *
   * Function that finds the overlapping range in 2 ranges
   * Return the new range to outputRange
   *
   **********************************************************************/
```

```
boolean GetOverlapRange (Range inputRange1,
      Range inputRange2, Range outputRange) {
   boolean success = false;

   outputRange.lower = Math.max(inputRange1.lower, inputRange2.lower) ;
   outputRange.upper = Math.min(inputRange1.upper, inputRange2.upper) ;

   if (outputRange.upper >= outputRange.lower) success = true;

   return success;
}


 /*********************************************************************
 *
 * Function that compares a permit range against a deny range
 * Saves the 2 possible output ranges to outputRange1 and outputRange2
 *
 * Return value signifies the extent of the overlap
 *
 *********************************************************************/
int GetPermitRange (Range permitRange, Range denyRange,
      Range outputRange1, Range outputRange2) {
   int numberOfOutputRanges = 0;
   /***********************************
    * 0 means no outputRange used, full overlap
    * 1 means only 1 outputRange used, there is overlap
    * 2 means 2 outputRanges used, there is overlap
    * -1 means no intersection or overlap
    ***********************************/

   /*********************************************
    * Determine 1st output range
    *********************************************/
   outputRange1.lower = permitRange.lower ;
   outputRange1.upper = Math.min(denyRange.lower-1 , permitRange.upper);
   if (outputRange1.upper >= outputRange1.lower)
            numberOfOutputRanges++;

   /*********************************************
    * Determine 2nd output range
    *********************************************/
   outputRange2.lower = Math.max(permitRange.lower, denyRange.upper + 1)
;
   outputRange2.upper = permitRange.upper ;
   if (outputRange2.upper >= outputRange2.lower)
```

```
                numberOfOutputRanges++;
    if (numberOfOutputRanges==1) outputRange1 = outputRange2 ;

    /*********************************************
    * If permit and deny ranges do not overlap
    *********************************************/
    if ( (denyRange.upper < permitRange.lower) ||
          (denyRange.lower > permitRange.upper) )
       numberOfOutputRanges = -1;

    return numberOfOutputRanges;
}

 /*********************************************************************
  *
  * Function that converts an IP address and a mask
  *     into an integer(long) range
  *
  * Takes an IP in dotted decimal format
  * Simplified IP address and mask handling,
  *     where lower value is set to inputIP
  *     and upper value is inputIP with full inputMask
  *
  * Currently assumes that router config uses Cisco recommendation of
  *     a contiguous range of IP addresses
  *
  * To add functionality later : to handle mask fully,
  *     for cases like "10.10.9.0", "0.0.4.255"
  *     where 9 = 1001, 4 = 100; 1001 and 1101 ; 13 = 1101
  *     and 10.10.9.0 with mask of 0.0.1.255,
  *     which actually refers to a range of 10.10.8.0 to 10.10.9.255
  *
  *********************************************************************/
Range convertIPtoIntegerRange (String inputIP, String inputMask) {
    Range output = new Range();
    int ipOctet1, ipOctet2, ipOctet3, ipOctet4;
    int maskOctet1, maskOctet2, maskOctet3, maskOctet4;
    int upperOctet1, upperOctet2, upperOctet3, upperOctet4 ;
    int lowerOctet1, lowerOctet2, lowerOctet3, lowerOctet4 ;
    String upperOctetString1, upperOctetString2,
          upperOctetString3, upperOctetString4 ;
    String lowerOctetString1, lowerOctetString2,
          lowerOctetString3, lowerOctetString4 ;

    /*****************************************************
     * If there are no IP addresses, the range is set as the max
```

```
 ****************************************************/
if (inputIP==null || inputIP.equalsIgnoreCase("any")) {
   output.lower = minIntegerIP ;
   output.upper = maxIntegerIP ;
}
else {
   /*****************************************************
    * Process IP address first
    * Break up IP address into octets
    *****************************************************/
   Scanner s = new Scanner(inputIP).useDelimiter("\\.");
   ipOctet1 = Integer.parseInt(s.next());
   ipOctet2 = Integer.parseInt(s.next());
   ipOctet3 = Integer.parseInt(s.next());
   ipOctet4 = Integer.parseInt(s.next());

  /*****************************************************
   * Calculate integer value of the IP address
   *****************************************************/
   output.lower = ( ipOctet1 * octet1Multiplier ) +
                  ( ipOctet2 * octet2Multiplier ) +
                  ( ipOctet3 * octet3Multiplier ) +
                  ( ipOctet4 * octet4Multiplier ) ;

   /*****************************************************
    * Process wildcard mask next
    * If there is no mask, single point in range
    *****************************************************/
   if (inputMask==null) {
      /* Disabled this section's functionality till
         full mask functionality is implemented
      // Calculate lower value of the IP address
      output.lower = ( ipOctet1 * octet1Multiplier ) +
                     ( ipOctet2 * octet2Multiplier ) +
                     ( ipOctet3 * octet3Multiplier ) +
                     ( ipOctet4 * octet4Multiplier ) ;
       */
      output.upper = output.lower;
   }
   else {
      /*****************************************************
       *  Break up wildcard mask into octets
       *****************************************************/
      s = new Scanner(inputMask).useDelimiter("\\.");
      maskOctet1 = Integer.parseInt(s.next());
      maskOctet2 = Integer.parseInt(s.next());
```

```
        maskOctet3 = Integer.parseInt(s.next());
        maskOctet4 = Integer.parseInt(s.next());

        /* Disabled this section's functionality till
           full mask functionality is implemented

        // Calculate lower value of IP range in string notation
        lowerOctetString1 = Integer.toString(ipOctet1 ^ maskOctet1) ;
        lowerOctetString2 = Integer.toString(ipOctet2 ^ maskOctet2) ;
        lowerOctetString3 = Integer.toString(ipOctet3 ^ maskOctet3) ;
        lowerOctetString4 = Integer.toString(ipOctet4 ^ maskOctet4) ;

        // Convert lower value of IP range from string to integer
        lowerOctet1 = Integer.parseInt(lowerOctetString1);
        lowerOctet2 = Integer.parseInt(lowerOctetString2);
        lowerOctet3 = Integer.parseInt(lowerOctetString3);
        lowerOctet4 = Integer.parseInt(lowerOctetString4);

        // Calculate integer value of the IP address
        output.lower = ( lowerOctet1 * octet1Multiplier ) +
                       ( lowerOctet2 * octet2Multiplier ) +
                       ( lowerOctet3 * octet3Multiplier ) +
                       ( lowerOctet4 * octet4Multiplier ) ;

         */

        /*** Calculate upper value of IP range in string ***/
        upperOctetString1 = Integer.toString(ipOctet1 | maskOctet1) ;
        upperOctetString2 = Integer.toString(ipOctet2 | maskOctet2) ;
        upperOctetString3 = Integer.toString(ipOctet3 | maskOctet3) ;
        upperOctetString4 = Integer.toString(ipOctet4 | maskOctet4) ;

        /*** Convert upper value of IP range from string to integer ***/
        upperOctet1 = Integer.parseInt(upperOctetString1);
        upperOctet2 = Integer.parseInt(upperOctetString2);
        upperOctet3 = Integer.parseInt(upperOctetString3);
        upperOctet4 = Integer.parseInt(upperOctetString4);

        /*** Calculate integer value of the IP address ***/
        output.upper = ( upperOctet1 * octet1Multiplier ) +
                       ( upperOctet2 * octet2Multiplier ) +
                       ( upperOctet3 * octet3Multiplier ) +
                       ( upperOctet4 * octet4Multiplier ) ;
    } // end of else statement on inputMask
} // end of else statement on inputIP
```

```
    return output;
}

/*************************************************************************
 *
 *   Function to pad a binary string with leading zeros.
 *   Input : binary (string)
 *   Output : 8-bit padded (string)
 *
 *************************************************************************/
String BinaryPadder (String inputString) {
    for (int counter=inputString.length(); counter<octetLength; counter++)
        inputString = "0" + inputString;
    return inputString;
}

/*************************************************************************
 *
 *   Function to convert an IP from integer into dotted decimal format
 *
 *************************************************************************/
String convertIntegertoIP (long inputInteger) {
    String outputString = new String();
    long octet1, octet1Mod, octet2, octet2Mod, octet3, octet3Mod, octet4 ;
    String octet1String, octet2String, octet3String, octet4String ;

    octet1 = inputInteger / octet1Multiplier;
    octet1Mod = inputInteger % octet1Multiplier;
    octet2 = octet1Mod / octet2Multiplier;
    octet2Mod = octet1Mod % octet2Multiplier;
    octet3 = octet2Mod / octet3Multiplier;
    octet3Mod = octet2Mod % octet3Multiplier;
    octet4 = octet3Mod / octet4Multiplier;

    octet1String = Long.toString(octet1);
    octet2String = Long.toString(octet2);
    octet3String = Long.toString(octet3);
    octet4String = Long.toString(octet4);

    outputString = octet1String + "." + octet2String + "."
          + octet3String + "." + octet4String ;
    return outputString;
}


/*************************************************************************
```

```
 *
 *   Function to convert a port into a range
 *
 ************************************************************************/
Range convertPortToRange (String inputLower, String inputUpper) {
   Range outputRange = new Range();
   if (inputLower==null || inputLower.equalsIgnoreCase("any")) {
      outputRange.lower = minPort;
      outputRange.upper = maxPort;
   }
   else {
      outputRange.lower = Long.parseLong(inputLower);
      if (inputUpper==null) outputRange.upper = maxPort;
      else outputRange.upper = Long.parseLong(inputUpper);
   }
   return outputRange;
}


/************************************************************************
 *
 *   Function to convert a protocol into a range
 *
 ************************************************************************/
Range convertProtocolToRange (String inputLower, String inputUpper){
   Range outputRange = new Range();
   if (inputLower==null || inputLower.equalsIgnoreCase("any")) {
      outputRange.lower = minProtocol;
      outputRange.upper = maxProtocol;
   }
   else {
      outputRange.lower = Long.parseLong(inputLower);
      if (inputUpper==null) outputRange.upper = maxProtocol;
      else outputRange.upper = Long.parseLong(inputUpper);
   }
   return outputRange;
}


 /************************************************************************
 *
 *   Function to convert an ACL rule into a tuple format
 *
 ************************************************************************/
Tuple convertACLRuletoTuple (ACLrule inputACLrule) {
   Tuple outputTuple = new Tuple();
```

```
    outputTuple.sourceIP = convertIPtoIntegerRange
          (inputACLrule.source, inputACLrule.sourceWildcard);
    outputTuple.destinationIP = convertIPtoIntegerRange
          (inputACLrule.destination, inputACLrule.destinationWildcard);

    outputTuple.sourcePort = convertPortToRange
          (inputACLrule.sourcePortLower, inputACLrule.sourcePortUpper);
    outputTuple.destinationPort = convertPortToRange
          (inputACLrule.destinationPortLower,
          inputACLrule.destinationPortUpper);

    outputTuple.protocol = convertProtocolToRange
          (inputACLrule.protocolLower, inputACLrule.protocolUpper);

    return outputTuple;
}


/*************************************************************************
 *
 *   Function to carry out a Union operation on 2 PacketSets
 *
 *************************************************************************/
PacketSet Union (PacketSet inputPS1, PacketSet inputPS2) {
    boolean success = false;
    PacketSet outputPS = new PacketSet();
    if (inputPS1!=null) outputPS.tupleArray.addAll(inputPS1.tupleArray);
    if (inputPS2!=null) outputPS.tupleArray.addAll(inputPS2.tupleArray);

    success = OptimizePacketSet (outputPS);
    return outputPS;
}

/*************************************************************************
 *
 *   Function to carry out an Intersection operation on 2 PacketSets
 *
 * Assumption : All tuples in a PacketSet are distinct
 *      (i.e., the 5 ranges in a tuple do not all overlap)
 *      Put another way, tuples cannot be combined further.
 *
 *************************************************************************/
boolean Intersection (PacketSet inputPS1, PacketSet inputPS2,
        PacketSet outputPS) {
    boolean success=false;
```

```
    Tuple tupleOut;
    outputPS.tupleArray.clear(); // Empty the outputPS

    for (int i=0; i<inputPS1.tupleArray.size(); i++)
        for (int j=0; j<inputPS2.tupleArray.size(); j++) {
            tupleOut = new Tuple();
            if (IntersectTuple ((Tuple)inputPS1.tupleArray.get(i),
                    (Tuple)inputPS2.tupleArray.get(j),tupleOut)) {
                success = true;
                outputPS.tupleArray.add(tupleOut);
            }
        }
    OptimizePacketSet(outputPS);

    return success;
}


/*************************************************************************
 *
 *   Function to carry out an Intersection operation on 2 tuples
 *
 *************************************************************************/
boolean IntersectTuple (Tuple tuple1, Tuple tuple2, Tuple tuple3) {
    boolean success = false;

    if (IntersectRange(tuple1.sourceIP, tuple2.sourceIP,
            tuple3.sourceIP, "intersect"))
        if (IntersectRange(tuple1.sourcePort, tuple2.sourcePort,
                tuple3.sourcePort, "intersect"))
            if (IntersectRange(tuple1.destinationIP,
                    tuple2.destinationIP, tuple3.destinationIP, "intersect"))
                if (IntersectRange(tuple1.destinationPort,
                        tuple2.destinationPort, tuple3.destinationPort,
                        "intersect"))
                    if (IntersectRange(tuple1.protocol, tuple2.protocol,
                            tuple3.protocol, "intersect"))
                        success=true;

    return success;
}


/*************************************************************************
 *
 *   Function to carry out an Intersection operation on 2 ranges
```

```
 *
 *  2 switches : for union and intersection
 *      For intersection operation, find smaller matching range of
 *            range1 and range2
 *      For union operation, find widest matching range of range1 and range2
 *
 *  Update : union switch no longer used)
 *
 ************************************************************************/
boolean IntersectRange (Range inputRange1, Range inputRange2,
       Range outputRange, String operation) {
    boolean success = false;
    String unionOp = "union";
    String intersectOp = "intersect";
    long lower, upper;

    /*****************************************************************
     * If the switch is for an intersection operation
     *****************************************************************/
    if (operation.equals(intersectOp)) {
        if (inputRange1.lower < inputRange2.lower)
            outputRange.lower = inputRange2.lower;
        else outputRange.lower = inputRange1.lower;

        if (inputRange1.upper < inputRange2.upper)
            outputRange.upper = inputRange1.upper;
        else outputRange.upper = inputRange2.upper;

        if (outputRange.upper < outputRange.lower) success=false;
        else success=true;
    }
    /*****************************************************************
     * If the switch is for a union operation
     *****************************************************************/
    else if (operation.equals(unionOp)){
        if (inputRange1.lower < inputRange2.lower) {
            outputRange.lower = inputRange1.lower;
            lower = inputRange2.lower;
        }
        else {
            outputRange.lower = inputRange2.lower;
            lower = inputRange1.lower;
        }

        if (inputRange1.upper < inputRange2.upper) {
            outputRange.upper = inputRange2.upper;
```

```
            upper = inputRange1.upper;
        }
        else {
            outputRange.upper = inputRange1.upper;
            upper = inputRange2.upper;
        }

        if (upper + 1 >= lower) success=true;//{

    }
    return success;
}


/*************************************************************************
 *
 *   Determine the reachability between 2 nodes in a network
 *
 *  Handles Reachability Upper Bound computations
 *
 *************************************************************************/
PacketSet InitializePath (NetworkConfig network, int source,
        int destination, PacketSet RLB) {
    int networkSize = network.tableOfRouters.size() ;
    /***********************************************
     * RUB Section Initialization
     ***********************************************/
    PacketSet finalPacketSet[][] = new PacketSet[networkSize][networkSize];
    PacketSet tempPacketSet[][] = new PacketSet[networkSize][networkSize];
    PacketSet intersectedPacketSet = new PacketSet();
    PacketSet output = new PacketSet();
    // Get the keys from tableOfRouters
    Enumeration routerList = network.tableOfRouters.keys();
    String router[] = new String[networkSize];

    /* Store router names in a lookup array
           that can be referenced by numbers */
    for (int i=0 ; i < networkSize; i++) {
        router[i] = (String) routerList.nextElement();
        System.out.println(i + " : " + router[i]); // Display router list
    }

    /* Initialize finalPacketSet[i][j] for all i */
    for (int i=0 ; i < networkSize; i++) {
        finalPacketSet[i][destination] = new PacketSet();
```

```
    /* (1) Check whether i and destination are neighbors on all interfaces
       (2) Find the packetset */
    finalPacketSet[i][destination] = InitializePacketSetRUB(
            (RouterConfig) network.tableOfRouters.get(router[i]),
            (RouterConfig) network.tableOfRouters.get(router[destination]));
}

/* Start algorithm to calculate reachability */
for (int m=0 ; m<networkSize-2; m++) {

    /* For each router i */
    for (int i=0 ; i<networkSize; i++) {

        /* if i==destination, jump to next iteration of for loop */
        if (i==destination) continue;

        tempPacketSet[i][destination] = new PacketSet();

        /* Initialize variables for router i */
        RouterConfig routerI = new RouterConfig();
        routerI = (RouterConfig) network.tableOfRouters.get(router[i]);
        PacketSet packetSetI = new PacketSet();

        /* For each interface on router i */
        Enumeration interfaceListI =
              routerI.tableOfInterfaceByIPs.elements();
        while (interfaceListI.hasMoreElements()) {

            InterfaceConfig interfaceOnI =
                  (InterfaceConfig) interfaceListI.nextElement();

            /* for each neighbor on the interface */
            for (int counterNeighbor=0;
               counterNeighbor<interfaceOnI.neighbors.size();
               counterNeighbor++){

               String neighborOfI = (String)
                  interfaceOnI.neighbors.get(counterNeighbor);

               /* for all k */
               for (int k=0; k<networkSize; k++) {

                  RouterConfig routerK = (RouterConfig)
                     network.tableOfRouters.get(router[k]);
                  /* for each interface on k */
                  Enumeration interfaceListK =
```

```
                routerK.tableOfInterfaceByIPs.keys();
            while (interfaceListK.hasMoreElements()) {
                String nextInterfaceOnK = (String)
                    interfaceListK.nextElement();
                if (nextInterfaceOnK.equalsIgnoreCase
                        (neighborOfI)) {

                    /* if k has an interface that is a neighbor
                     *   if i, get the intersecting
                     *   PacketSet  */
                    InterfaceConfig interfaceOnK =
                            new InterfaceConfig();
                    interfaceOnK = (InterfaceConfig)
                    routerK.tableOfInterfaceByIPs.get(nextInterfaceOnK);

                    /* Find the intersecting PacketSet
                       of router i and k */
                    PacketSet packetSetIK = new PacketSet();
                    packetSetIK = GetPacketSetOverLink
                            (routerI, interfaceOnI,
                            routerK, interfaceOnK);

                    /* Reachability calculations */
                    intersectedPacketSet = new PacketSet();
                    Intersection (packetSetIK,
                            finalPacketSet[k][destination],
                            intersectedPacketSet);

                    tempPacketSet[i][destination] =
                            Union (tempPacketSet[i][destination],
                            intersectedPacketSet) ;
                }
            }
        } // end of for (int k=0; k<networkSize; k++)

      } // end of for ()
    } // end of while (interfaceListI.hasMoreElements())
    /*****************************************************
     * Assign values to RUB from i to destination
     *****************************************************/

    finalPacketSet[i][destination].tupleArray.clear();
    finalPacketSet[i][destination].tupleArray.addAll
            (tempPacketSet[i][destination].tupleArray);

} // end of for (int i=0 ; i<networkSize; i++)
```

```
    } // end of for (int m=0 ; m<networkSize-2; m++)

    /* Prepare results of reachability analysis for output */
    output = finalPacketSet[source][destination];

    return output;
}


/**************************************************************************
 *
 *  Initialize packetSetRUB from router1 to router2
 *
 *  check whether 2 routers are neighbors on any interface
 *  find the intersecting packetset on the link
 *  from router1 to router2
 *
 **************************************************************************/
PacketSet InitializePacketSetRUB (RouterConfig router1,
        RouterConfig router2) {
    PacketSet outputPS = new PacketSet();
    PacketSet packetSet1 = new PacketSet();
    PacketSet packetSet2 = new PacketSet();
    PacketSet intersectedPS = new PacketSet();

    // if router1==router2, set outputPS to full set (i.e., 1)
    if (router1==router2) outputPS = NoFilters();
    // Determine resultant PacketSet if neighbors found
    // No neighbor ==> empty PacketSet
    else {

        // Check for neighbor relation
        Enumeration interfaceList1 = router1.tableOfInterfaceByIPs.elements();
        // Look at one interface at a time on router1
        while (interfaceList1.hasMoreElements()) {
            InterfaceConfig interface1 = (InterfaceConfig)
                    interfaceList1.nextElement();

            // Check through list of neighbors on interface, one by one
            for (int count=0; count<interface1.neighbors.size();count++){
                String neighbor = (String) interface1.neighbors.get(count);
                Enumeration interfaceList2 = router2.tableOfInterfaceByIPs.keys();

                // Check through IPs of interfaces on router2, one by one
                while (interfaceList2.hasMoreElements()) {
                    String interface2IP = (String) interfaceList2.nextElement();
```

```
                    // if both interfaces on the routers are neighbors
                    if (interface2IP.equalsIgnoreCase(neighbor)) {

                        InterfaceConfig interface2 = (InterfaceConfig)
                            router2.tableOfInterfaceByIPs.get(interface2IP);
                        intersectedPS = GetPacketSetOverLink
                                (router1, interface1, router2, interface2);
                        outputPS.tupleArray.addAll(intersectedPS.tupleArray);
                    } // endif
                }// endwhile
            }// endfor
        }// endwhile
        OptimizePacketSet (outputPS);
    } // endelse
    return outputPS;
}

/*************************************************************************
 *
 *  Find the PacketSet over a link
 *      from the outbound queue of the 1st interface
 *      to the inbound queue of the 2nd interface
 *
 *************************************************************************/
PacketSet GetPacketSetOverLink (RouterConfig fromRouter,
        InterfaceConfig fromInterface, RouterConfig toRouter,
        InterfaceConfig toInterface) {
    PacketSet outputPS = new PacketSet();
    PacketSet inboundPacketSet = new PacketSet();
    PacketSet outboundPacketSet = new PacketSet();

    outboundPacketSet  =  GetPacketSetonInterface(fromRouter,  fromInterface,
false);
    inboundPacketSet = GetPacketSetonInterface(toRouter, toInterface, true);
    Intersection(inboundPacketSet, outboundPacketSet, outputPS);

    return outputPS;
}


/*************************************************************************
 *
 * Determine which are the PacketSets used by one interface
 * on either the inbound or outbound queue
 *
```

```
 ***************************************************************************/
PacketSet GetPacketSetonInterface (RouterConfig inputRouter,
        InterfaceConfig inputInterface, boolean inOrOut) {
    /* If inOrOut is true ==> inbound queue
     * If inOrOut is false ==> outbound queue
     */
    PacketSet PS = new PacketSet();
    ArrayList filters = new ArrayList();
    if (inOrOut) filters.addAll(inputInterface.inFilters);
    else filters.addAll(inputInterface.outFilters);

    if (filters.size()==0) PS = NoFilters();
    else {
        PacketSet tempPS = new PacketSet();
        try {
            tempPS = (PacketSet)
                        inputRouter.mapOfPacketSets.get(filters.get(0));
            PS.tupleArray.addAll(tempPS.tupleArray);
        } catch (Exception e) {
            System.err.println("ERROR: Router " + inputRouter.hostName
                        + " may not contain ACL #" + filters.get(0) +
                         " that " + inputInterface.interfaceName
                        + " has specified !");
        }

        if (filters.size()>1)
            for (int i=1; i<filters.size(); i++) {
                PacketSet interimPS = new PacketSet();
                interimPS.tupleArray.addAll(PS.tupleArray);
                try {
                    tempPS = (PacketSet)
                                inputRouter.mapOfPacketSets.get(filters.get(i));
                } catch (Exception e) {
                    System.err.println("ERROR: Router " + inputRouter.hostName +
                            " may not contain ACL #" + filters.get(0) +
                            " that " + inputInterface + " has specified !");
                }
                PS.tupleArray.clear();
                Intersection (interimPS, tempPS, PS);
            }
    }

    return PS;
}

/***************************************************************************
```

```
     *
     *  Set default value if no filters are applied to an interface
     *  Default value is to permit any
     *
     ***************************************************************************/
    PacketSet NoFilters() {
        PacketSet output = new PacketSet();
        Tuple out = new Tuple();
        out.sourceIP.lower = minIntegerIP;
        out.sourceIP.upper = maxIntegerIP;
        out.destinationIP.lower = minIntegerIP;
        out.destinationIP.upper = maxIntegerIP ;
        out.sourcePort.lower = minPort;
        out.sourcePort.upper = maxPort;
        out.destinationPort.lower = minPort;
        out.destinationPort.upper = maxPort;
        out.protocol.lower = minProtocol;
        out.protocol.upper = maxProtocol;
        output.tupleArray.add(out);
        return output;
    }

}
```

H. PARSER.JAVA

```
package StaticReachabilityAnalysis;
/*
 * Parser.java
 *
 * Created          :   July 25, 2006, 3:20 AM
 * Last Modified    :   December 12, 2006
 * Author           :   Eric Gregory Wong
 *
 * ********** MAIN CALLING FUNCTION FOR PARSING AND PACKETSET CREATION **********
 *
 * Functionality:
 *
 * 1. Parses Router Config files with commands:
 *      a. Router hostname
 *      b. Interface:
 *          i.  Name
 *          ii. IP address and mask
 *          iii.Interface access-group in and out
 *          iv. ***Exception - ignores the above if keyword "remark" is used
 *      c. Access-list:
 *          i.  Standard lists
 *          ii. Extended lists with:
 *              (1) ACL #
 *              (2) Dynamic name
 *              (3) Timeout minutes         (to be added)
 *              (4) Permit or deny
 *              (5) Protocol
 *              (6) Source + wildcard
 *              (7) Destination + wildcard
 *              (8) Precedence              (to be added)
 *              (9) TOS                     (to be added)
 *              (10) Log                    (to be added)
 */

import java.io.*;
import java.util.*;

class Parser {
    /***************
     * Debugging file names
     * These files will be saved in the source code or project directory when run
     ***************/
    String testFilename1 = "Test File 1 Parsed output check direct from read.txt";
    String testFilename2 = "Test File 2 Intermediate parser check.txt";
```

```
    String testFilename3 = "Test File 3 Last ACL Output.txt";
    String testFilename4 = "Test File 4 Hashtable Output (Post).txt";
    String testFilename5 = "Test File 5 Hashtable Output (from read).txt";
    String testFilename6 = "Test File 6 Last Interface check.txt";
    String testFilename7 = "Test File 7 Files in Folder Check.txt";

    /*** Creation of a Parser class ***/
    Parser(NetworkConfig network, File inputDir, File outputDir) throws
IOException {

        /**************************************************************************
           Initialize test files that will be output to the project directory
        **************************************************************************/
        FileWriter outputTestFile1 = new FileWriter(new File(testFilename1));
        FileWriter outputTestFile2 = new FileWriter(new File(testFilename2));
        PrintWriter outputTestFile3 = new PrintWriter (new BufferedWriter
              (new FileWriter(testFilename3)));
        PrintWriter outputTestFile4 = new PrintWriter (new BufferedWriter
              (new FileWriter(testFilename4)));
        FileWriter outputTestFile5 = new FileWriter(new File(testFilename5));
        PrintWriter outputTestFile6 = new PrintWriter (new BufferedWriter
              (new FileWriter(testFilename6)));
        FileWriter outputTestFile7 = new FileWriter(new File(testFilename7));

        /**************************************************************************
            Parse the files in the input directory one at a time
        **************************************************************************/
        String filenames[] = inputDir.list();
        network.networkName = inputDir.getName() ;
        for(int n=0;n<filenames.length;n++) {
           outputTestFile7.write(filenames[n] + "\r\n");
           File inputFile = new File(inputDir,filenames[n]);

           /**************************************************************************
              Set up ACLs, Router Interfaces and Routers
           **************************************************************************/
           String currentACLnumber = new String();
           String previousACLnumber = new String();
           LinkedList acl = new LinkedList();
           RouterConfig router = new RouterConfig();
           InterfaceConfig routerInterface = new InterfaceConfig();

           /**************************************************************************
              Set up a Scanner to read the file using tokens
           **************************************************************************/
           Scanner scanner = null;
```

```
try {
   scanner = new Scanner (inputFile);
   scanner.useDelimiter("\n");
   //scanner.useDelimiter(System.getProperty("line.separator"));
        // doesn't work for .conf files
} catch (FileNotFoundException e) {
    System.out.println ("File not found!"); // for debugging
    System.exit (0); // Stop program if no file found
}

/*************************************************************************
     Read each token in every scanned line
*************************************************************************/
boolean interfaceFlag = false; // to check if interface is being processed
/* Read line by line */
while (scanner.hasNext()) {
   /* Read token by token in each line */
   Scanner lineScanner = new Scanner(scanner.next());
   String keyword;

   if (lineScanner.hasNext()) {
      keyword = lineScanner.next();

       /*****************************************************************
          This section handles the hostname
       *****************************************************************/
      if (keyword.equals("hostname")){
         router.hostName=lineScanner.next();
         outputTestFile1.write(router.hostName + "\r\n"); // for debugging
      }
      /*****************************************************************
         This section handles the interface table creation
      *****************************************************************/
      else if (keyword.equals("interface")) {
         /* Create a new interface object */
         routerInterface = new InterfaceConfig();

         /* Get the interface name */
         routerInterface.interfaceName = lineScanner.next();

         outputTestFile1.write(routerInterface.interfaceName
                + "\r\n"); // for debugging
         interfaceFlag = true;
         continue;
      }
```

```
/*************************************************************************
   This section handles the interface configuration section
*************************************************************************/
else if (interfaceFlag) {
   /* End of interface section is denoted by ! */
   if (keyword.equals("!")) {
      interfaceFlag = false;
      /*************************************************
      * Store interface objects in Router Config
      *************************************************/
      /* Store object in table with name as the key */
      if (routerInterface.interfaceName !=null) {
         /* there may be interface that didn't specify ip address
          specify here "no ip address"
          to prevent a null pointer error ***/
         if (routerInterface.ipAddress == null)  {
            routerInterface.ipAddress = "no ip address";
         }
         router.tableOfInterfaceByNames.put
               (routerInterface.interfaceName, routerInterface);
      }
      /* Store object in table with ip as the key
        If no ip, object will not be stored */
      if (routerInterface.ipAddress != null &&
            !routerInterface.ipAddress.equals("no ip address") )
         router.tableOfInterfaceByIPs.put
               (routerInterface.ipAddress, routerInterface);
      continue;
   }
   /* Handle lines that begin with "ip" */
   else if (keyword.equals("ip")) {
      outputTestFile1.write (keyword + " | "); // for debugging
      String IP_argument_1, IP_argument_2, IP_argument_3;
      IP_argument_1 = lineScanner.next();
      if (IP_argument_1.equals("address") ||
                  IP_argument_1.equals("access-group")){
         IP_argument_2 = lineScanner.next();
         IP_argument_3 = lineScanner.next();

         if (IP_argument_1.equals ("address")){
            routerInterface.ipAddress = IP_argument_2;
            routerInterface.ipMask = IP_argument_3;
         }
         else if (IP_argument_1.equals ("access-group")) {
            if (IP_argument_3.equals("in"))
               routerInterface.inFilters.add(IP_argument_2);
```

```
            else if (IP_argument_3.equals("out"))
               routerInterface.outFilters.add(IP_argument_2);
         }
         outputTestFile1.write (keyword + " | " + IP_argument_1 + " | "
            + IP_argument_2 + " | " + IP_argument_3 + "\r\n" );
                  // for debugging
      }
   } // End of else if (keyword.equals("ip"))

   /* Handles interfaces with no ip specified, keyword "no" */
   else if (keyword.equals("no")) {
      String nextArgument = lineScanner.next();
      if (nextArgument.equals("ip")) {
         nextArgument = lineScanner.next();
         if (nextArgument.equals("address")) {
            routerInterface.ipAddress = "no ip address";
            outputTestFile1.write (keyword + " | no ip address + \r\n");
         }
      }
   } // End of else if (keyword.equals("no"))
} // end of else if (interfaceFlag)

/************************************************************
  This section handles ACL rules that start with "access-list"
************************************************************/
else if (keyword.equals("access-list")) {
 labelBreakHere: {
     int i = 0;
     ACLrule aclRule = new ACLrule();
     aclRule.accessList = keyword;

   /*************************************
    * read access-list into argument array
    * Although only 18 needed,
    *  allocated 12 extra to read options
    *  not handled
    *************************************/
   String[] argument = new String[30] ;

   while (lineScanner.hasNext()) {
      argument[i] = lineScanner.next();
      if (argument[i].equals ("!")) break;
      i++;
   }

   // Test Function 1 : output argument array into a test file
```

```
// for checking
outputTestFile1.write (keyword + " | ");
for (int j=0; j<i; j++) outputTestFile1.write (argument[j] + " | ");
// end of Test Function 1

/*************************************************************************
   Parse argument array into correct ACL rule structure
*************************************************************************/
if (argument[0].equals ("rate-limit")) break labelBreakHere;
       // Skip this section if "rate-limit" found

aclRule.accessListNumber = argument[0];
currentACLnumber = argument[0];

// Initialize previousACLnumber
if (previousACLnumber.length()==0) previousACLnumber =
        currentACLnumber;

// Check whether this is a new ACL
if (currentACLnumber.equals(previousACLnumber)) ; // do nothing
else {
// if different ACL number, store the previous ACL in router config
   router.tableOfACLs.put(previousACLnumber, acl);
   outputTestFile5.write("\r\nPrevious ACL #" +
        previousACLnumber
         + "\r\nCurrent ACL #" + currentACLnumber + "\r\n");
   outputTestFile5.write(acl + "\r\n");
   acl = new LinkedList();
   previousACLnumber = currentACLnumber;
}

int k = 1;
    // k  is a position marker used to check the rest of the ACL

/*************************************************************************
   Parse the rest of the ACL rule
*************************************************************************/
if (argument[k].equals ("remark") ) { // No need to parse
                          // further if ACL contains "remark"
   aclRule.remark=true;
   k=i;
}

while (k<i) {
// check whether ACL is permit or deny
   boolean checkPermitDeny = false;
```

```
// track whether this is a standard IP Access List
boolean standardIPAccessList = false ;
// track whether this is an extended IP Access List
 boolean extendedIPAccessList = false ;

/*************************************************************************
    This section handles the Dynamic part in an ACL rule
*************************************************************************/
if (argument[k].equals ("dynamic")) {
    aclRule.dynamic="dynamic";
    k++;
    aclRule.dynamicName=argument[k];
}

/*************************************************************************
    This section handles the Timeout part in an ACL rule
*************************************************************************/

     // timeout section to be added

/*************************************************************************
    This section handles the Permit and Deny keywords
     in an ACL rule
*************************************************************************/
if (argument[k].equals ("permit")) {
    aclRule.permitDeny="permit";
    k++;
    checkPermitDeny = true;
}
else if (argument[k].startsWith("deny")) {
                // .equals doesn't work here
    aclRule.permitDeny="deny";
    k++;
    checkPermitDeny = true;
}

/*************************************************************************
    This section handles the Protocol, Source
    and Destination parts in an ACL rule
    Entry into this section only after "Permit" or "Deny" found
*************************************************************************/
 if (checkPermitDeny) {
    // check whether ACL is standard or extended
    if (Integer.valueOf(currentACLnumber)>=1 &&
          Integer.valueOf(currentACLnumber)<=99 )
              standardIPAccessList=true ;
```

```
else if (Integer.valueOf(currentACLnumber)>=100 &&
        Integer.valueOf(currentACLnumber)<=199 )
            extendedIPAccessList=true;
else if (Integer.valueOf(currentACLnumber)>=1300 &&
        Integer.valueOf(currentACLnumber)<=1999 )
            standardIPAccessList=true ;
else if (Integer.valueOf(currentACLnumber)>=2000 &&
        Integer.valueOf(currentACLnumber)<=2699 )
            extendedIPAccessList=true;

/*****************************************
   Process standard ACL rules
*****************************************/
if (standardIPAccessList) {
   aclRule.source = argument[k]; //store source IP
   // Process Source Mask
   if (k+1<i) {
      if (argument[k+1].contains(".") ||
            argument[k+1].contains("any")) {
         k++;
         aclRule.sourceWildcard = argument[k];
      }
   }
} // end of one line of Standard ACL parsing

/*****************************************
   Process extended ACL rules
*****************************************/
else if (extendedIPAccessList) {
   aclRule.protocolLower = GetProtocolNumber
         (argument[k]);
   if (aclRule.protocolLower.equals("256")) {
      aclRule.protocolLower = "0";
      aclRule.protocolUpper = "255";
   }
   else aclRule.protocolUpper = aclRule.protocolLower;
   k++;

   /*****************************************
      Process source fields
   *****************************************/
   if (argument[k].equals("any")) {
      aclRule.source = argument[k];
      k++;
      k = ParsePort (aclRule, argument, k, "source");
   }
```

```
        else if (argument[k].equals("host")) {
            k++;
            aclRule.source = argument[k];
            k++;
            k = ParsePort (aclRule, argument, k, "source");
        }
        else {
            aclRule.source = argument[k];
            k++;
            aclRule.sourceWildcard = argument[k];
            k++;
            k = ParsePort (aclRule, argument, k, "source");
        }

        /*****************************************
             Process destination fields
        *****************************************/
        /*** If the destination keyword is "any" ***/
        if (argument[k].equals("any")) {
            aclRule.destination = argument[k];
            k++;
            if (argument[k]!=null)
                k = ParsePort (aclRule, argument, k, "destination");
        }
        else if (argument[k].equals("host")) {
            k++;
            aclRule.destination = argument[k];
            k++;
            if (argument[k]!=null)
                k = ParsePort (aclRule, argument, k, "destination");
        }
        else {
            aclRule.destination = argument[k];
            k++;
            aclRule.destinationWildcard = argument[k];
            k++;
            if (argument[k]!=null)
                k = ParsePort (aclRule, argument, k, "destination");
        }
    } // end of one line of Extended ACL parsing
} // End of protocol, source and destination parts

/*************************************************************************
    This section handles the precedence, tos and log parts
     in an ACL rule
*************************************************************************/
```

```
                // to be added
                // end of precedence, tos and log parts

                k++;
              }

              new DebugTools().IntermediateParserCheck(aclRule,
                    outputTestFile2); // for debugging
              outputTestFile1.write ("\r\n" ); // for debugging

              // Add current ACL line to the Linked List
              // if it is not meant as a remark
              if (aclRule.remark==false) acl.add(aclRule);

            } // end of labelBreakHere
          } // End of section handling ACLS that start with "access-list"
        }

    } // end of while (scanner.hasNext())

    /*************************************************
     * Check whether this is a new ACL
     *************************************************/
    if (currentACLnumber.equals(previousACLnumber)) {
      router.tableOfACLs.put(previousACLnumber, acl);
      outputTestFile5.write("\r\n" + previousACLnumber + "\r\n"
            + currentACLnumber + "\r\n"); // for debugging
      outputTestFile5.write(acl + "\r\n"); // for debugging
    }

    outputTestFile4.println(router);
    outputTestFile6.println(routerInterface);
    new DebugTools().CheckLastACLOutput(acl,outputTestFile3);
    // System.out.println(router.hostName); // for debugging

    /*********************************************************************
          Save the router to the tableOfRouters object
          in the network, using its hostname as the key
    *********************************************************************/
    if (router.hostName!=null) network.tableOfRouters.put
          (router.hostName, router);

} // end of for loop search files in folder

/*** Close all the debugging/test files ***/
```

```
        outputTestFile1.close();
        outputTestFile2.close();
        outputTestFile3.close();
        outputTestFile4.close();
        outputTestFile5.close();
        outputTestFile6.close();
        outputTestFile7.close();

        /*************************************************************************
           Determine which routers are the neighbors of which
        *************************************************************************/
        boolean successProcessNeighbors =
                    ProcessNeighbors(network.tableOfRouters);

        /*************************************************************************
           Create packet sets for all the routers in the network
        *************************************************************************/
        new PacketSet().CreateAllPacketSets (network);

        /*************************************************************************
           Send the network data to the output directory
        *************************************************************************/
        new NetworkDataDump(network, outputDir);

        /*************************************************************************
           Notify user the parsing has been completed
        *************************************************************************/
        System.out.println("\r\n\r\n----> Successful completion of parsing <----\r\n");
    } // end of Parser


    /*****************************************************************************
     *
     *   Function to determine the neighbors of routers in a network.
     *    Checks the interface on each router against the interface on every other
router.
     *   Store the IPs of each router's neighbors in its respective interface objects.
     *
     *****************************************************************************/
    boolean ProcessNeighbors (Hashtable HT1) {
        boolean success=false;
        Hashtable interfaceTable1, interfaceTable2;
        int routerCounter = 0;
        RouterConfig currentRouter;
        Enumeration routerList = HT1.elements();
        try {
```

```
while ( routerList.hasMoreElements() ) {
   currentRouter = (RouterConfig) routerList.nextElement();
   System.out.println("------- Processing Neighbors ------  Router : "
         + currentRouter.hostName) ; // for debugging
   routerCounter ++;

   Enumeration routerList2 = HT1.elements();
   RouterConfig routerToCompare = null;

   //skip some routers
   for (int i = 0; i < routerCounter; i++) {
      if (routerList2.hasMoreElements()) routerToCompare =
            (RouterConfig) routerList2.nextElement();
   }

   while (routerList2.hasMoreElements()) {
      routerToCompare = (RouterConfig) routerList2.nextElement();
      Enumeration interfaceList =
               currentRouter.tableOfInterfaceByNames.elements();

      while (interfaceList.hasMoreElements()) {
         InterfaceConfig currentInterface =
               (InterfaceConfig) interfaceList.nextElement();
         Enumeration interfaceList2 =
               routerToCompare.tableOfInterfaceByNames.elements();

         while (interfaceList2.hasMoreElements()) {

            InterfaceConfig interfaceToCompare =
                  (InterfaceConfig) interfaceList2.nextElement();
            String prefix1, prefix2 ;
            if (currentInterface.ipAddress.equals("no ip address") ||
                  interfaceToCompare.ipAddress.equals("no ip address")) {
               success=false;
            }
            else if (currentInterface.ipAddress.equals
                  (interfaceToCompare.ipAddress)) {
               // do nothing
            }
            else if  (currentInterface.ipMask.equals
                (interfaceToCompare.ipMask)) {
               prefix1 = GetPrefix (currentInterface.ipAddress,
                     currentInterface.ipMask);
               prefix2 = GetPrefix (interfaceToCompare.ipAddress,
                     interfaceToCompare.ipMask);
               if (prefix1.equals(prefix2)) { // they are neighbors
```

```
currentInterface.neighbors.add(interfaceToCompare.ipAddress);

interfaceToCompare.neighbors.add(currentInterface.ipAddress);
                                success=true;
                             }
                          }
                       } // end of while (interfaceList2.hasMoreElements())
                    } // end of while (interfaceList.hasMoreElements())
                 } // end of while (routerList2.hasMoreElements())
              } // end of while ( routerList.hasMoreElements() )
           } catch (Exception e) { System.out.println ("Error - " + e); }
           return success;
        }


        /*******************************************************************************
         *
         *   Function to calculate the network prefix of an IP address.
         *   Input : IP address (string), IP mask (string)
         *   Output : Network prefix (string)
         *
         *******************************************************************************/
        String GetPrefix (String ip, String mask) {
           String Prefix;
           int ipOctet1=0, ipOctet2=0, ipOctet3=0, ipOctet4=0;
           int maskOctet1=0, maskOctet2=0, maskOctet3=0, maskOctet4=0;

           /*** Break up ip string into octets ***/
           Scanner s = new Scanner(ip).useDelimiter("\\.");
           ipOctet1 = Integer.parseInt(s.next());
           ipOctet2 = Integer.parseInt(s.next());
           ipOctet3 = Integer.parseInt(s.next());
           ipOctet4 = Integer.parseInt(s.next());

           /*** Break up mask string into octets ***/
           s = new Scanner(mask).useDelimiter("\\.");
           maskOctet1 = Integer.parseInt(s.next());
           maskOctet2 = Integer.parseInt(s.next());
           maskOctet3 = Integer.parseInt(s.next());
           maskOctet4 = Integer.parseInt(s.next());

           /*** Determine the network prefix based on the ip and mask ***/
           Prefix = Integer.toString(ipOctet1 & maskOctet1) + "." +
                    Integer.toString(ipOctet2 & maskOctet2) + "." +
                    Integer.toString(ipOctet3 & maskOctet3) + "." +
```

```
                Integer.toString(ipOctet4 & maskOctet4);

        return Prefix;
    }


    /*****************************************************************************
     *
     *   Function to convert ports in an ACL rule to its IANA number assignment.
     *   Input : ACL rule (ACLrule), parsed argument (string), position k (int),
     *           port name or number (string)
     *   Output : position k, which is one position after reading the port arguments
     *    The upper and lower port numbers are stored in the ACL rule that was
input.
     *   Limitation: Does not handle neq operator yet
     *
     *****************************************************************************/
    int ParsePort (ACLrule aclRule, String[] argument, int k, String port) {

        // handles eq, gt, lt, range
        // to include handling for neq
        if (argument[k].equalsIgnoreCase("eq")) { // handles eq
            k++;
            if (port.equals("source")) {
                aclRule.sourcePortLower = GetPortNumber (argument[k]);
                aclRule.sourcePortUpper = aclRule.sourcePortLower;
                k++;
            }
            else if (port.equals("destination")) {
                aclRule.destinationPortLower = GetPortNumber (argument[k]);
                aclRule.destinationPortUpper = aclRule.destinationPortLower;
            }
        }
        else if (argument[k].equalsIgnoreCase("gt")) {
            k++;
            if (port.equals("source")) {
                aclRule.sourcePortLower = GetPortNumber (argument[k]);
                k++;
            }
            else if (port.equals("destination"))
                aclRule.destinationPortLower = GetPortNumber (argument[k]);
        }
        else if (argument[k].equalsIgnoreCase("lt")) {
            k++;
            if (port.equals("source")) {
                aclRule.sourcePortUpper = GetPortNumber (argument[k]);
```

```
            k++;
        }
        else if (port.equals("destination"))
            aclRule.destinationPortUpper = GetPortNumber (argument[k]);
    }
    else if (argument[k].equalsIgnoreCase("range")) {
        k++;
        if (port.equals("source")) {
            aclRule.sourcePortLower = GetPortNumber (argument[k]);
            k++;
            aclRule.sourcePortUpper = GetPortNumber (argument[k]);
            k++;
        }
        else if (port.equals("destination")) {
            aclRule.destinationPortLower = GetPortNumber (argument[k]);
            k++;
            aclRule.destinationPortUpper = GetPortNumber (argument[k]);
        }
    }
    return k;
}

/*******************************************************************************
 *
 *   Function to lookup the IANA number assignment of a port.
 *   Input : port name or number (string)
 *   Output : port number (string)
 *
 *******************************************************************************/
String GetPortNumber (String port) {
    String portNumber;
    // Note switch statement does not work with strings
    if (port.equalsIgnoreCase("tcpmux")) portNumber = "1";
    else if (port.equalsIgnoreCase("ftp-data")) portNumber = "20";
    else if (port.equalsIgnoreCase("ftp")) portNumber = "21";
    else if (port.equalsIgnoreCase("ssh")) portNumber = "22";
    else if (port.equalsIgnoreCase("telnet")) portNumber = "23";
    else if (port.equalsIgnoreCase("smtp")) portNumber= "25";
    else if (port.equalsIgnoreCase("dsp")) portNumber= "33";
    else if (port.equalsIgnoreCase("time")) portNumber= "37";
    else if (port.equalsIgnoreCase("rap")) portNumber= "38";
    else if (port.equalsIgnoreCase("rlp")) portNumber= "39";
    else if (port.equalsIgnoreCase("name")) portNumber= "42";
    else if (port.equalsIgnoreCase("nameserver")) portNumber= "42";
    else if (port.equalsIgnoreCase("nicname")) portNumber= "43";
    else if (port.equalsIgnoreCase("dns")) portNumber = "53";
```

```
    else if (port.equalsIgnoreCase("domain")) portNumber = "53";
    else if (port.equalsIgnoreCase("bootps")) portNumber = "67";
    else if (port.equalsIgnoreCase("bootpc")) portNumber = "68";
    else if (port.equalsIgnoreCase("tftp")) portNumber = "69";
    else if (port.equalsIgnoreCase("gopher")) portNumber = "70";
    else if (port.equalsIgnoreCase("finger")) portNumber = "79";
    else if (port.equalsIgnoreCase("http")) portNumber = "80";
    else if (port.equalsIgnoreCase("www")) portNumber = "80";
    else if (port.equalsIgnoreCase("kerberos")) portNumber = "88";
    else if (port.equalsIgnoreCase("pop2")) portNumber = "109";
    else if (port.equalsIgnoreCase("pop3")) portNumber = "110";
    else if (port.equalsIgnoreCase("sunrpc")) portNumber = "111";
    else if (port.equalsIgnoreCase("ident")) portNumber = "113";
    else if (port.equalsIgnoreCase("auth")) portNumber = "113";
    else if (port.equalsIgnoreCase("sftp")) portNumber = "115";
    else if (port.equalsIgnoreCase("nntp")) portNumber = "119";
    else if (port.equalsIgnoreCase("netbios-ns")) portNumber = "137";
    else if (port.equalsIgnoreCase("netbios-dg")) portNumber = "138";
    else if (port.equalsIgnoreCase("netbios-ss")) portNumber = "139";
    else if (port.equalsIgnoreCase("sqlsrv")) portNumber = "156";
    else if (port.equalsIgnoreCase("snmp")) portNumber = "161";
    else if (port.equalsIgnoreCase("bgp")) portNumber = "179";
    else if (port.equalsIgnoreCase("exec")) portNumber = "512";
    else if (port.equalsIgnoreCase("shell")) portNumber = "514";
    else portNumber = port;
    return portNumber;
}

/*******************************************************************************
 *
 *   Function to lookup the IANA number assignment of a protocol.
 *   Input : protocol name or number (string)
 *   Output : protocol number (string)
 *
 *******************************************************************************/
String GetProtocolNumber (String protocol) {
    String protocolNumber;
    if(protocol.equalsIgnoreCase("icmp")) protocolNumber = "1" ;
    else if(protocol.equalsIgnoreCase("igmp")) protocolNumber = "2" ;
    //else if(protocol.equalsIgnoreCase("ip")) protocolNumber = "4" ;
    else if(protocol.equalsIgnoreCase("ip")) protocolNumber = "256" ;
        // special case to indicate all protocols. No actual protocol number 256.
    else if(protocol.equalsIgnoreCase("tcp")) protocolNumber = "6" ;
    else if(protocol.equalsIgnoreCase("egp")) protocolNumber = "8" ;
    else if(protocol.equalsIgnoreCase("igp")) protocolNumber = "9" ;
    else if(protocol.equalsIgnoreCase("udp")) protocolNumber = "17" ;
```

```
        else if(protocol.equalsIgnoreCase("rdp")) protocolNumber = "27" ;
        else if(protocol.equalsIgnoreCase("ipv6")) protocolNumber = "41" ;
        else if(protocol.equalsIgnoreCase("rsvp")) protocolNumber = "46" ;
        else if(protocol.equalsIgnoreCase("eigrp")) protocolNumber = "88" ;
        else if(protocol.equalsIgnoreCase("l2tp")) protocolNumber = "115" ;
        else protocolNumber = protocol ;
        return protocolNumber;
    }

} //Parser
```

I. PARSERGUI.JAVA

```
package StaticReachabilityAnalysis;
/*
 * ParserGUI.java
 *
 * Main Class File
 *
 * Created          :   November 18, 2006
 * Last Modified    :   November 30, 2006
 * Author           :   Eric Gregory Wong
 */

import java.io.*;
import javax.swing.*;

public class ParserGUI extends javax.swing.JFrame {
    NetworkConfig network;
    String outputDirString;
    String programVersion = "Version 1.0";

    /** Creates new form ParserGUI */
    public ParserGUI() {

        initComponents();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc=" Generated Code ">
    private void initComponents() {
        jPopupMenu1 = new javax.swing.JPopupMenu();
        jPanel2 = new javax.swing.JPanel();
        jButton1 = new javax.swing.JButton();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jSeparator1 = new javax.swing.JSeparator();
        jButton2 = new javax.swing.JButton();
        jLabel3 = new javax.swing.JLabel();
        jTextField1 = new javax.swing.JTextField();
        jButton3 = new javax.swing.JButton();
        jLabel4 = new javax.swing.JLabel();
        jTextField2 = new javax.swing.JTextField();
        jButton4 = new javax.swing.JButton();
```

```
        jLabel6 = new javax.swing.JLabel();
        jButton5 = new javax.swing.JButton();
        jLabel7 = new javax.swing.JLabel();
        jLabel5 = new javax.swing.JLabel();
        jButton6 = new javax.swing.JButton();

        org.jdesktop.layout.GroupLayout jPanel2Layout = new
org.jdesktop.layout.GroupLayout(jPanel2);
        jPanel2.setLayout(jPanel2Layout);
        jPanel2Layout.setHorizontalGroup(

jPanel2Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(0, 100, Short.MAX_VALUE)
        );
        jPanel2Layout.setVerticalGroup(

jPanel2Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(0, 100, Short.MAX_VALUE)
        );


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setBackground(new java.awt.Color(255, 255, 255));
        jButton1.setText("Parse Now");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });
        jButton1.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                jButton1MouseClicked(evt);
            }
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                jButton1MouseReleased(evt);
            }
        });

        jLabel1.setFont(new java.awt.Font("Arial", 0, 12));
        jLabel1.setText("Step 1 : Type or browse for the source and destination
directories");
        jLabel1.addPropertyChangeListener(new
java.beans.PropertyChangeListener() {
            public void propertyChange(java.beans.PropertyChangeEvent evt) {
                jLabel1PropertyChange(evt);
            }
```

```
});

jLabel2.setFont(new java.awt.Font("Arial", 1, 28));
jLabel2.setText("STATIC REACHABILITY ANALYSIS TOOLKIT");

jButton2.setText("Calculate Reachability Bounds");
jButton2.setEnabled(false);
jButton2.setOpaque(false);
jButton2.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton2ActionPerformed(evt);
    }
});
jButton2.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        jButton2MouseClicked(evt);
    }
});

jLabel3.setFont(new java.awt.Font("Arial", 0, 16));
jLabel3.setText("Source Directory :");

jTextField1.setFont(new java.awt.Font("Arial", 0, 14));
jTextField1.setText("D:\\Routers");
jTextField1.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        jTextField1MouseClicked(evt);
    }
});

jButton3.setText("Browse");
jButton3.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        jButton3MouseClicked(evt);
    }
});

jLabel4.setFont(new java.awt.Font("Arial", 0, 16));
jLabel4.setText("Destination Directory :");

jTextField2.setFont(new java.awt.Font("Arial", 0, 14));
jTextField2.setText("D:\\Output");
jTextField2.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        jTextField2MouseClicked(evt);
    }
```

```
        });

        jButton4.setText("Browse");
        jButton4.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                jButton4MouseClicked(evt);
            }
        });

        jLabel6.setFont(new java.awt.Font("Arial", 0, 12));
        jLabel6.setText("Step 2 : Click on \"Parse Now\" to begin parsing the router configuration files in the source directory");
        jLabel6.addPropertyChangeListener(new java.beans.PropertyChangeListener() {
            public void propertyChange(java.beans.PropertyChangeEvent evt) {
                jLabel6PropertyChange(evt);
            }
        });

        jButton5.setText("Quit");
        jButton5.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton5ActionPerformed(evt);
            }
        });

        jLabel7.setIcon(new javax.swing.ImageIcon("D:\\NPS\\Thesis\\Static Reachability Analysis Program\\Static Reachability Analysis\\src\\StaticReachabilityAnalysis\\NPS-logo.jpg"));
        jLabel7.setVerticalAlignment(javax.swing.SwingConstants.TOP);

        jLabel5.setFont(new java.awt.Font("Arial", 0, 12));
        jLabel5.setText("Attention: existing contents of destination directory will be deleted when parsing starts");

        jButton6.setText("About");
        jButton6.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                jButton6MouseClicked(evt);
            }
        });

        org.jdesktop.layout.GroupLayout layout = new org.jdesktop.layout.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
```

```
        layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
        .add(layout.createSequentialGroup()

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                .add(layout.createSequentialGroup()
                    .add(30, 30, 30)

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                        .add(layout.createSequentialGroup()
                            .add(jLabel7,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,                    138,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                            .add(jLabel2))
                        .add(layout.createSequentialGroup()

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                            .add(jLabel3)
                            .add(jLabel4))
                        .add(26, 26, 26)

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                            .add(layout.createSequentialGroup()
                                .add(jTextField1,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,                    408,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                                .add(23, 23, 23)
                                .add(jButton3))
                            .add(layout.createSequentialGroup()
                                .add(jTextField2,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,                    408,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                                .add(23, 23, 23)
                                .add(jButton4))
                            .add(jLabel5,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 599, Short.MAX_VALUE)))))
                .add(layout.createSequentialGroup()
                    .add(20, 20, 20)

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.TRAILING,
false)
                        .add(org.jdesktop.layout.GroupLayout.LEADING,          jLabel1,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
```

```
                    .add(org.jdesktop.layout.GroupLayout.LEADING, jLabel6,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 603,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)))
                .add(layout.createSequentialGroup()
                    .addContainerGap()
                    .add(jSeparator1,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 799,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)))
                .addContainerGap())
            .add(org.jdesktop.layout.GroupLayout.TRAILING,
layout.createSequentialGroup()
                .addContainerGap(228, Short.MAX_VALUE)
                .add(jButton1)
                .add(80, 80, 80)
                .add(jButton2)
                .add(145, 145, 145)

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.TRAILING,
false)
                .add(jButton5,      org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .add(jButton6,      org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                .add(47, 47, 47))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(layout.createSequentialGroup()

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                .add(layout.createSequentialGroup()
                    .add(30, 30, 30)
                    .add(jLabel7,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 104,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                .add(layout.createSequentialGroup()
                    .add(67, 67, 67)
                    .add(jLabel2)))
                .add(49, 49, 49)

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.TRAILING)
                .add(jLabel4)
                .add(layout.createSequentialGroup()

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                    .add(jButton3)
```

```
                        .add(jTextField1,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                        .add(jLabel3))
                    .add(39, 39, 39)

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                        .add(jTextField2,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                        .add(jButton4))))
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(jLabel5)

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                    .add(layout.createSequentialGroup()
                        .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED,  33,
Short.MAX_VALUE)

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                        .add(jButton1)
                        .add(jButton2))
                    .add(30, 30, 30))
                .add(layout.createSequentialGroup()
                    .add(17, 17, 17)
                    .add(jButton6)
                    .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                    .add(jButton5)
                    .add(17, 17, 17)))
                .add(jSeparator1,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(jLabel1)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(jLabel6)
                .addContainerGap(org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );
        pack();
    }// </editor-fold>

    private void jButton1MouseReleased(java.awt.event.MouseEvent evt) {
```

```
// TODO add your handling code here:
        jLabel1.setText("Parsing in progress ......");
        jLabel6.setText(" ");
    }

    private void jButton6MouseClicked(java.awt.event.MouseEvent evt) {
// TODO add your handling code here:
            JFrame frame2 = new JFrame();
            JOptionPane.showMessageDialog(frame2,"Developed by Eric Wong and
" +
                    "Geoffrey Xie, \r\nDepartment of Computer Science,\r\n" +
                    "Naval Postgraduate School,\r\nNovember 2006.\r\n - " +
                    programVersion, "About this tool",
JOptionPane.INFORMATION_MESSAGE);
    }

    private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
        System.exit(0);
    }

    private void jLabel6PropertyChange(java.beans.PropertyChangeEvent evt) {
// TODO add your handling code here:
    }

    private void jButton4MouseClicked(java.awt.event.MouseEvent evt) {
// TODO add your handling code here:
        String filename = jTextField2.getText();
        JFileChooser chooser = new JFileChooser(new File(filename));
        chooser.setApproveButtonText("Ok");
        chooser.setApproveButtonToolTipText("Press OK after selecting a source
directory");
        chooser.setBackground(java.awt.Color.white);
        chooser.setCurrentDirectory(new java.io.File("D:\\"));

chooser.setFileSelectionMode(javax.swing.JFileChooser.DIRECTORIES_ONLY)
;

        // Show open dialog; this method does not return until the dialog is closed:
        int result = chooser.showOpenDialog(this);
        //int result = jFileChooser1.showOpenDialog(this);

        // Determine which button was clicked to close the dialog:
        switch (result) {
            case JFileChooser.APPROVE_OPTION:
                File selFile = chooser.getSelectedFile();
```

```
            jTextField2.setText(selFile.getAbsolutePath());
            break;
        case JFileChooser.CANCEL_OPTION:
            // Cancel or the close-dialog icon was clicked:
            break;
        case JFileChooser.ERROR_OPTION:
            // The selection process did not complete successfully:
            break;
    }
  }

  private void jTextField2MouseClicked(java.awt.event.MouseEvent evt) {
// TODO add your handling code here:
  }

  private void jButton3MouseClicked(java.awt.event.MouseEvent evt) {
// TODO add your handling code here:
    String filename = jTextField1.getText();
    JFileChooser chooser = new JFileChooser(new File(filename));
    chooser.setApproveButtonText("Ok");
    chooser.setApproveButtonToolTipText("Press OK after selecting a source
directory");
    chooser.setBackground(java.awt.Color.white);
    chooser.setCurrentDirectory(new java.io.File("D:\\"));

chooser.setFileSelectionMode(javax.swing.JFileChooser.DIRECTORIES_ONLY)
;

    // Show open dialog; this method does not return until the dialog is closed:
    int result = chooser.showOpenDialog(this);
    //int result = jFileChooser1.showOpenDialog(this);

    // Determine which button was clicked to close the dialog:
    switch (result) {
        case JFileChooser.APPROVE_OPTION:
            File selFile = chooser.getSelectedFile();
            jTextField1.setText(selFile.getAbsolutePath());
            break;
        case JFileChooser.CANCEL_OPTION:
            // Cancel or the close-dialog icon was clicked:
            break;
        case JFileChooser.ERROR_OPTION:
            // The selection process did not complete successfully:
            break;
    }
  }
```

```
    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
        setVisible(false);
        new PathChooser(network, outputDirString).setVisible(true);
    }

    private void jTextField1MouseClicked(java.awt.event.MouseEvent evt) {
// TODO add your handling code here:
    }

    private void jButton2MouseClicked(java.awt.event.MouseEvent evt) {
// TODO add your handling code here:
        //if (jButton2.)

    }

    private void jLabel1PropertyChange(java.beans.PropertyChangeEvent evt) {
// TODO add your handling code here:

    }

    private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {
// TODO add your handling code here:
        try {
            String input = jTextField1.getText();
            File inputDir = new File(input);
            String output = jTextField2.getText();
            outputDirString = output;
            File outputDir = new File(output);
            network = new NetworkConfig();
            new Parser(network, inputDir, outputDir);
            jLabel1.setText("Parsing completed and network files saved to " +
output);
            jLabel6.setText("Step 3 : Click on Calculate Reachability Bounds");
            jButton2.setEnabled(true);
        } catch (Exception e) {
            System.out.println ("Program Error - " + e);
            JFrame frame = new JFrame();
            JOptionPane.showMessageDialog(frame,"Error - " + e,
                    "Program Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
```

```
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new ParserGUI().setVisible(true);

            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JButton jButton3;
    private javax.swing.JButton jButton4;
    private javax.swing.JButton jButton5;
    private javax.swing.JButton jButton6;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JPopupMenu jPopupMenu1;
    private javax.swing.JSeparator jSeparator1;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextField jTextField2;
    // End of variables declaration

}
```

J. PATHCHOOSER.JAVA

```
package StaticReachabilityAnalysis;
/*
 * PathChooser.java
 *
 * Created on November 18, 2006, 1:35 AM
 */

/**
 *
 * @author  user
 */
import java.util.*;
import java.io.*;
import javax.swing.*;

public class PathChooser extends javax.swing.JFrame {
    NetworkConfig theNetwork = new NetworkConfig();
    String theOutputDir;

    /** Creates new form PathChooser */
    public PathChooser(NetworkConfig network, String outputDir) {

        initComponents();
        /* Populate the 2 combo boxes with the list of routers parsed earlier */
        Enumeration routerList = network.tableOfRouters.keys();
        String router[] = new String[network.tableOfRouters.size()];
        for (int i=0 ; i < network.tableOfRouters.size(); i++) {
            router[i] = (String) routerList.nextElement();
            jComboBox1.addItem(router[i]);
            jComboBox2.addItem(router[i]);
        }
        if (network.tableOfRouters.size()>1) jComboBox2.setSelectedIndex(1);
        theNetwork = network;
        theOutputDir = outputDir;
        try{
                String errorFileName = "Errors Encountered.txt";
                File errorFile = new File (theOutputDir, errorFileName);
                PrintStream errFile = new PrintStream (errorFile);
                System.setErr(errFile);
        } catch (Exception e) { System.out.println ("Error - " + e); }

    }

    /** This method is called from within the constructor to
     * initialize the form.
```

```
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc=" Generated Code ">
    private void initComponents() {
        jDialog1 = new javax.swing.JDialog();
        jDialog2 = new javax.swing.JDialog();
        jComboBox1 = new javax.swing.JComboBox();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jComboBox2 = new javax.swing.JComboBox();
        jButton1 = new javax.swing.JButton();
        jSeparator1 = new javax.swing.JSeparator();
        jLabel4 = new javax.swing.JLabel();
        jButton2 = new javax.swing.JButton();

        org.jdesktop.layout.GroupLayout jDialog1Layout = new
org.jdesktop.layout.GroupLayout(jDialog1.getContentPane());
        jDialog1.getContentPane().setLayout(jDialog1Layout);
        jDialog1Layout.setHorizontalGroup(

jDialog1Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(0, 400, Short.MAX_VALUE)
        );
        jDialog1Layout.setVerticalGroup(

jDialog1Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(0, 300, Short.MAX_VALUE)
        );
        org.jdesktop.layout.GroupLayout jDialog2Layout = new
org.jdesktop.layout.GroupLayout(jDialog2.getContentPane());
        jDialog2.getContentPane().setLayout(jDialog2Layout);
        jDialog2Layout.setHorizontalGroup(

jDialog2Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(0, 400, Short.MAX_VALUE)
        );
        jDialog2Layout.setVerticalGroup(

jDialog2Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(0, 300, Short.MAX_VALUE)
        );

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
```

```
        jLabel1.setText("Source Router");

        jLabel2.setText("Destination Router");

        jLabel3.setFont(new java.awt.Font("Arial", 1, 18));
        jLabel3.setText("REACHABILITY CALCULATION");

        jButton1.setText("Calculate It !");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });
        jButton1.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                jButton1MouseClicked(evt);
            }
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                jButton1MouseReleased(evt);
            }
        });

        jLabel4.setText("Select the source and destination routers for path
calculation, then click \"Calculate It!\"");

        jButton2.setText("Quit");
        jButton2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton2ActionPerformed(evt);
            }
        });

        org.jdesktop.layout.GroupLayout layout = new
org.jdesktop.layout.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jSeparator1, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 598,
Short.MAX_VALUE)
            .add(layout.createSequentialGroup()
                .addContainerGap()
                .add(jLabel4, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 578,
Short.MAX_VALUE)
                .addContainerGap())
            .add(layout.createSequentialGroup()
                .add(38, 38, 38)
```

```
.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                .add(jLabel1)
                .add(jLabel2))
            .add(69, 69, 69)

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING,
false)
                .add(jComboBox2,                                                0,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .add(jComboBox1, 0, 347, Short.MAX_VALUE))
            .addContainerGap(54, Short.MAX_VALUE))
        .add(org.jdesktop.layout.GroupLayout.TRAILING,
layout.createSequentialGroup()
            .addContainerGap(165, Short.MAX_VALUE)

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                .add(layout.createSequentialGroup()
                    .add(jButton1)
                    .add(102, 102, 102)
                    .add(jButton2))
                .add(jLabel3))
            .add(163, 163, 163))
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
        .add(org.jdesktop.layout.GroupLayout.TRAILING,
layout.createSequentialGroup()
            .add(24, 24, 24)
            .add(jLabel3)
            .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED,          43,
Short.MAX_VALUE)

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                .add(jLabel1)
                .add(jComboBox1,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
            .add(20, 20, 20)

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                .add(jLabel2)
                .add(jComboBox2,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
```

```
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                .add(50, 50, 50)

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                    .add(jButton1)
                    .add(jButton2))
                .add(36, 36, 36)
                .add(jSeparator1,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,                                 10,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(jLabel4,       org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
14, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );
        pack();
    }// </editor-fold>

    private void jButton1MouseReleased(java.awt.event.MouseEvent evt) {
// TODO add your handling code here:
        jLabel4.setText("Reachability calculations in progress ......");
        // disable the "Calculate It" button while calculations are ongoing
        jButton1.setEnabled(false);
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:

    }

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
        System.exit(0);
    }

    private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {
// TODO add your handling code here:
        ComputeNow();
        jButton1.setEnabled(true);
    }

    /**************************************************************************
     *
     * Run the reachability computation code
     *
```

```
     ***************************************************************************/
    void ComputeNow(){
        // Path Calculations
        int source = jComboBox1.getSelectedIndex();
        int destination = jComboBox2.getSelectedIndex();
        boolean successComputeNow=false;
        boolean successFileSave=false;
        /*********************************************
         * If the source and destination routers selected are the same, pop up error
dialog box
         * Reachability computations will not be run
         *********************************************/
        if (source==destination) {
            JFrame frame = new JFrame();
            JOptionPane.showMessageDialog(frame,
                    "Source and destination routers must not be the same !",
                    "Selection Error", JOptionPane.ERROR_MESSAGE);
        }
        else {
            String sourceName = (String) jComboBox1.getSelectedItem();
            String destinationName = (String) jComboBox2.getSelectedItem();
            PacketSet rUB = new PacketSet();
            PacketSet rLB = new PacketSet();
            /*************************
             * Display reachability calculation information on default
                    system.out display
             *************************/
            System.out.println("======= Calculating Reachability =======");
            rUB = rUB.InitializePath(theNetwork,source, destination, rLB);
            System.out.println("\r\nReachability Upper Bound from " + source +
                    " to " + destination + " : " + rUB);
            System.out.println("-----------------------------------------------------" +
                    "---------\r\n");

            /*************************
             * Save reachability calculation results to a file in the output directory
             *************************/
            String outputFileName = "Reachability from " + sourceName
                    + " to " + destinationName + ".txt";
            try {
                File outputFile = new File (theOutputDir, outputFileName);
                PrintWriter outFile = new PrintWriter (outputFile);
                outFile.println("Reachability Upper Bound from " + sourceName + " to "
                        + destinationName + " : " + "\r\n");
                outFile.println(rUB);
                /********************* RLB Section disabled
```

```
            outFile.println();
            outFile.println("-------------------------------------------------");
            outFile.println("Reachability Lower Bound from " + sourceName + " to "
                + destinationName + " : " + "\r\n");
            outFile.println(rLB);
             */
            outFile.close();
            successFileSave = true;
          } catch (Exception e) { System.out.println ("Error - " + e); }

          successComputeNow = true;
        }
        /**********************
         * Display status of reachability calculations in GUI
         **********************/
        if (successComputeNow && successFileSave)
          jLabel4.setText("Reachability calculations completed and saved to "
                  + theOutputDir);
        else if (successComputeNow)
          jLabel4.setText("Reachability calculations completed" +
                          "but could not be saved to " + theOutputDir);
        else jLabel4.setText("Reachability calculations failed. Select the source " +
            "and destination routers for path calculation, then click Calculate It!");

    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
/*      java.awt.EventQueue.invokeLater(new Runnable() {
          public void run() {
            new PathChooser().setVisible(true);
          }
        });
 */
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JComboBox jComboBox1;
    private javax.swing.JComboBox jComboBox2;
    private javax.swing.JDialog jDialog1;
    private javax.swing.JDialog jDialog2;
    private javax.swing.JLabel jLabel1;
```

```
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JSeparator jSeparator1;
    // End of variables declaration

}
```

K. RANGE.JAVA

```
package StaticReachabilityAnalysis;
/*
 * Range.java
 *
 */

import java.util.*;

class Range {

    /*String lower;
    String upper;*/
    long lower;
    long upper;

    /** Creates a new instance of Range */
    Range() {
        /*lower = "";
        upper = "";*/
        lower = -1L;
        upper = -1L;
    }
    public String toString () {
        return "[ " + lower + " , " + upper + " ] " ;
    }
}
```

L. ROUTERCONFIG.JAVA

```
package StaticReachabilityAnalysis;
/*
 * RouterConfig.java
 *
 * Created on August 16, 2006, 1:20 AM
 *
 * Last modified August 22, 2006
 *
 * Contains the router configuration class
 */

import java.util.*;
import java.io.*;

class RouterConfig {
    String hostName; // Router name
    Hashtable tableOfInterfaceByNames; // Table of pointers to Interface objects
                                //    keyed by the interface names
    Hashtable tableOfInterfaceByIPs; // Table of pointers to Interface objects
                               //    keyed by the interface IPs
    Hashtable tableOfACLs; // Table of pointers to ACLs
    TreeMap mapOfPacketSets; // Map of packet filters in tuple-form
          // Use the ACL numbers as keys to point to the packetfilter objects
(tuples)

    /** Creates a new instance of RouterConfig */
    RouterConfig() {
        hostName = null;
        tableOfInterfaceByNames = new Hashtable();
        tableOfInterfaceByIPs = new Hashtable();
        tableOfACLs = new Hashtable();
        mapOfPacketSets = new TreeMap(new TComp());
        //Interface = null; // doesn't work if set to null --> it's a pointer!
    }

    // not used???
    public String toString() {
        return hostName + " "
                + tableOfInterfaceByNames + " "
                + tableOfInterfaceByIPs + " "
                + tableOfACLs + " "
                + mapOfPacketSets + " ";
    }
```

```
    // Do a dump of the RouterConfig object for debugging/checking purposes
    void Debug(RouterConfig a, File outputDir) {
      try {
        String outputFileName = "Router Dump -
".concat(a.hostName.concat(".txt"));
        File Output = new File (outputDir, outputFileName);
        PrintWriter out = new PrintWriter (new BufferedWriter
                (new FileWriter(Output)));

        // Set up debugger object exactly like RouterConfig class
        // This eases checking that the debugger handles everything in the class
        String debugHostName = a.hostName;
        Hashtable debugTableOfInterfaceByNames = a.tableOfInterfaceByNames;
        Hashtable debugTableOfInterfaceByIPs = a.tableOfInterfaceByIPs;
        Hashtable debugTableOfACLs = a.tableOfACLs;
        TreeMap debugMapOfPacketSets = a.mapOfPacketSets ;

        ACLrule debugACLrule;
        LinkedList debugACL;

        InterfaceConfig debugInterfaces;


        out.println("Router Config Debugger");
        out.println("---------------------\r\n");
        out.println("Host name       : " + debugHostName + "\r\n");
        out.println("-----------------------------------------------" +"\r\n");

        String acl;
        Set set = debugMapOfPacketSets.entrySet();
        Iterator aclSet = set.iterator();
        while (aclSet.hasNext()) {
           Map.Entry mapPS = (Map.Entry) aclSet.next();
           acl = (String) mapPS.getKey();
           out.println("ACL number      : " + acl);

           debugACL = (LinkedList) debugTableOfACLs.get(acl);
           int LL_size = debugACL.size();
           out.println("# of ACL rules  : " + LL_size);
           out.println("*************************");
           out.println(debugMapOfPacketSets.get(acl));
           int counter = 0;
           while(counter < LL_size) {
              debugACLrule = (ACLrule) debugACL.get(counter);
              out.println("   Keyword / ACL#  : " + debugACLrule.accessList
```

```
            + " / " + debugACLrule.accessListNumber);
        out.println("   Keyword / Dyn#  : " + debugACLrule.dynamic
            + " / " + debugACLrule.dynamicName);
        out.println("   Keyword / TOmin : " + debugACLrule.timeout
            + " / " + debugACLrule.timeoutMinutes);
        out.println("   Permit or Deny  : " + debugACLrule.permitDeny);
        out.println("   Protocol Range  : " + debugACLrule.protocolLower +
            " / " + debugACLrule.protocolUpper);
        out.println("   Src / Wildcard  : " + debugACLrule.source
            + " / " + debugACLrule.sourceWildcard);
        out.println("   Src Port Range  : " + debugACLrule.sourcePortLower +
            " / " + debugACLrule.sourcePortUpper);
        out.println("   Dest / Wildcard : " + debugACLrule.destination
             + " / " + debugACLrule.destinationWildcard );
        out.println("   Dest Port Range : "+ debugACLrule.destinationPortLower
            + " / " + debugACLrule.destinationPortUpper);
        out.println("   Keyword / Prec# : " + debugACLrule.precedenceKeyword
            + " / " + debugACLrule.precedence);
        out.println("   Keyword / TOS   : " + debugACLrule.tosKeyword
            + " / " + debugACLrule.tos);
        out.println("   Keyword / Log   : " + debugACLrule.logKeyword
            + " / " + debugACLrule.logInput);
        out.println("   ********************************");
        counter++;
    }
    //if (ACL_number.hasMoreElements())
    if (aclSet.hasNext())
        out.println("++++++++++++++++++++++++++++++++++++++++++++++\r\n");
    else out.println();
}

out.println("----------------------------------------------" +"\r\n");

// Objects within become a set?!!
String interfaceName;
Enumeration interfaceProperties = debugTableOfInterfaceByNames.keys();
while(interfaceProperties.hasMoreElements()) {
    interfaceName = (String) interfaceProperties.nextElement();
    out.println("Interface name : " + interfaceName);

    debugInterfaces = (InterfaceConfig)
            debugTableOfInterfaceByNames.get(interfaceName);
    out.println("IP address      : " + debugInterfaces.ipAddress);
    out.println("IP mask         : " + debugInterfaces.ipMask);
    out.println("Neighbors       : " + debugInterfaces.neighbors);
    out.println("In Filters      : " + debugInterfaces.inFilters);
```

```
            out.println("Out Filters     : " + debugInterfaces.outFilters);
            out.println();
        }

        out.println("-----------------------------------------------" +"\r\n");

        String Interface_IP;
        Enumeration interfaceProperties_2 = debugTableOfInterfaceByIPs.keys();
        while(interfaceProperties_2.hasMoreElements()) {
            Interface_IP = (String) interfaceProperties_2.nextElement();
            out.println("Interface IP   : " + Interface_IP);

            debugInterfaces                    =                    (InterfaceConfig)
debugTableOfInterfaceByIPs.get(Interface_IP);
            out.println("Interface name  : " + debugInterfaces.interfaceName);
            out.println("IP mask         : " + debugInterfaces.ipMask);
            out.println("Neighbors       : " + debugInterfaces.neighbors);
            out.println("In Filters      : " + debugInterfaces.inFilters);
            out.println("Out Filters     : " + debugInterfaces.outFilters);
            out.println();
        }

        out.close();
      } catch (Exception e) { System.out.println ("Error - " + e); }
    }

}
```

M. TCOMP.JAVA

```
package StaticReachabilityAnalysis;
/*
 * TComp.java
 *
 */

import java.util.*;

class TComp implements Comparator {

    // This Comparator overrides the default compare function
    // Used for the TreeMap in RouterConfig
    // Enables correct sorting of the ACL number string
    public int compare (Object a, Object b) {
        int aValue, bValue;

        aValue = Integer.valueOf((String) a) ;
        bValue = Integer.valueOf((String) b);
        // if a is smaller than b, then a will come first
        return (aValue-bValue);
    }

    /** Creates a new instance of TComp */
    TComp() {
    }

}
```

N. TUPLE.JAVA

```
package StaticReachabilityAnalysis;
/*
 * Tuple.java
 *
*/

import java.util.*;

class Tuple {

    Range sourceIP;
    Range sourcePort;
    Range destinationIP;
    Range destinationPort;
    Range protocol;

    /** Creates a new instance of Tuple */
    Tuple() {

        sourceIP = new Range();
        sourcePort = new Range();
        destinationIP = new Range();
        destinationPort = new Range();
        protocol = new Range();
    }

    public String toString () {
        PacketSet pS = new PacketSet();
        return "[ " + pS.convertIntegertoIP(sourceIP.lower) + " , " +
                pS.convertIntegertoIP(sourceIP.upper) + " ] ; " +
               "[ " + sourcePort.lower + " , " + sourcePort.upper + " ] ; " +
               "[ " + pS.convertIntegertoIP(destinationIP.lower) + " , " +
                pS.convertIntegertoIP(destinationIP.upper) + " ] ; " + "[ " +
                destinationPort.lower + " , " + destinationPort.upper + " ] ; " +
               "[ " + protocol.lower +  " , " + protocol.upper + " ] ";
    }

}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. PACKETSET EFFICIENCY ANALYSIS

Router #	Number of ACLs	Number of ACL rules	Number of PacketSets	% of PacketSets to ACL rules	Total ACL rules per router	Total PacketSets per router
1	0	0	0	0.0%	0	0
2	1	1	1	100.0%	1	1
3	44	3	1	33.3%	409	811
		7	6	85.7%		
		2	1	50.0%		
		6	6	100.0%		
		5	5	100.0%		
		2	1	50.0%		
		2	1	50.0%		
		13	9	69.2%		
		8	6	75.0%		
		2	4	200.0%		
		15	10	66.7%		
		2	2	100.0%		
		8	4	50.0%		
		7	6	85.7%		
		9	16	177.8%		
		5	6	120.0%		
		9	16	177.8%		
		9	6	66.7%		
		2	2	100.0%		
		3	2	66.7%		
		7	16	228.6%		
		22	68	309.1%		
		10	28	280.0%		
		7	17	242.9%		
		4	2	50.0%		
		8	20	250.0%		
		9	25	277.8%		
		4	3	75.0%		
		5	6	120.0%		
		13	40	307.7%		
		9	24	266.7%		
		15	42	280.0%		
		13	40	307.7%		
		14	42	300.0%		
		17	50	294.1%		
		9	24	266.7%		
		10	18	180.0%		
		23	80	347.8%		
		56	97	173.2%		

Router #	Number of ACLs	Number of ACL rules	Number of PacketSets	% of PacketSets to ACL rules	Total ACL rules per router	Total PacketSets per router
		7	16	228.6%		
		8	6	75.0%		
		8	19	237.5%		
		6	8	133.3%		
		6	10	166.7%		
4	12	3	1	33.3%	49	34
		5	5	100.0%		
		3	2	66.7%		
		3	2	66.7%		
		2	1	50.0%		
		13	9	69.2%		
		8	6	75.0%		
		2	2	100.0%		
		1	1	100.0%		
		1	1	100.0%		
		4	2	50.0%		
		4	2	50.0%		
5	11	5	5	100.0%	45	35
		1	1	100.0%		
		3	2	66.7%		
		2	1	50.0%		
		13	9	69.2%		
		8	6	75.0%		
		2	2	100.0%		
		2	4	200.0%		
		1	1	100.0%		
		4	2	50.0%		
		4	2	50.0%		
6	16	3	1	33.3%	90	76
		6	6	100.0%		
		5	5	100.0%		
		2	1	50.0%		
		13	9	69.2%		
		8	6	75.0%		
		15	10	66.7%		
		5	6	120.0%		
		4	3	75.0%		
		5	6	120.0%		
		2	4	200.0%		
		4	2	50.0%		
		4	2	50.0%		
		5	6	120.0%		
		4	3	75.0%		
		5	6	120.0%		

Router #	Number of ACLs	Number of ACL rules	Number of PacketSets	% of PacketSets to ACL rules	Total ACL rules per router	Total PacketSets per router
7	0	0	0	0.0%	0	0
8	10	3	1	33.3%	96	199
		5	5	100.0%		
		2	1	50.0%		
		13	9	69.2%		
		8	6	75.0%		
		13	31	238.5%		
		38	136	357.9%		
		4	2	50.0%		
		4	2	50.0%		
		6	6	100.0%		
9	8	3	1	33.3%	55	78
		5	5	100.0%		
		2	1	50.0%		
		13	9	69.2%		
		8	6	75.0%		
		16	52	325.0%		
		4	2	50.0%		
		4	2	50.0%		
10	12	1	1	100.0%	68	321
		3	1	33.3%		
		3	3	100.0%		
		6	6	100.0%		
		4	2	50.0%		
		2	2	100.0%		
		2	1	50.0%		
		14	126	900.0%		
		14	126	900.0%		
		16	50	312.5%		
		1	1	100.0%		
		2	2	100.0%		
11	1	1	1	100.0%	1	1
12	1	1	1	100.0%	1	1
13	8	3	1	33.3%	60	84
		5	5	100.0%		
		2	1	50.0%		
		13	9	69.2%		
		8	6	75.0%		
		21	58	276.2%		
		4	2	50.0%		
		4	2	50.0%		
14	4	3	1	33.3%	21	17
		5	5	100.0%		
		4	3	75.0%		

Router #	Number of ACLs	Number of ACL rules	Number of PacketSets	% of PacketSets to ACL rules	Total ACL rules per router	Total PacketSets per router
		9	8	88.9%		
15	1	3	1	33.3%	3	1
16	32	3	1	33.3%	208	345
		6	6	100.0%		
		5	5	100.0%		
		2	1	50.0%		
		13	9	69.2%		
		8	6	75.0%		
		4	12	300.0%		
		5	16	320.0%		
		6	16	266.7%		
		5	10	200.0%		
		5	10	200.0%		
		5	10	200.0%		
		5	10	200.0%		
		5	10	200.0%		
		9	11	122.2%		
		8	20	250.0%		
		5	10	200.0%		
		5	10	200.0%		
		7	2	28.6%		
		5	10	200.0%		
		5	10	200.0%		
		7	20	285.7%		
		5	10	200.0%		
		7	16	228.6%		
		10	28	280.0%		
		5	10	200.0%		
		1	1	100.0%		
		10	10	100.0%		
		5	10	200.0%		
		7	16	228.6%		
		23	24	104.3%		
		7	5	71.4%		
17	6	7	6	85.7%	65	86
		30	12	40.0%		
		6	6	100.0%		
		4	1	25.0%		
		10	24	240.0%		
		8	37	462.5%		
18	22	3	1	33.3%	82	69
		2	1	50.0%		
		2	2	100.0%		
		2	2	100.0%		

Router #	Number of ACLs	Number of ACL rules	Number of PacketSets	% of PacketSets to ACL rules	Total ACL rules per router	Total PacketSets per router
		5	5	100.0%		
		2	2	100.0%		
		3	2	66.7%		
		2	1	50.0%		
		13	9	69.2%		
		8	6	75.0%		
		4	3	75.0%		
		7	13	185.7%		
		4	3	75.0%		
		1	1	100.0%		
		1	1	100.0%		
		4	4	100.0%		
		4	2	50.0%		
		4	2	50.0%		
		4	3	75.0%		
		1	1	100.0%		
		2	2	100.0%		
		4	3	75.0%		
19	8	3	1	33.3%	52	60
		5	5	100.0%		
		2	1	50.0%		
		13	9	69.2%		
		8	6	75.0%		
		4	2	50.0%		
		4	2	50.0%		
		13	34	261.5%		
20	7	2	1	50.0%	26	26
		5	5	100.0%		
		4	2	50.0%		
		4	2	50.0%		
		4	2	50.0%		
		5	8	160.0%		
		2	6	300.0%		

TOTAL	**204**	**1332**	**2245**	**168.54%**	**1332**	**2245**
MEAN	**10.20**	**6.47**	**10.90**	**127.53%**	**66.60**	**112.25**
STD DEV	**11.30**	**6.23**	**19.89**	**115.04%**	**94.48**	**192.68**
MAX	**44.00**	**56.00**	**136.00**	**900.00%**	**409.00**	**811.00**
MIN	**0.00**	**0.00**	**0.00**	**0.00%**	**0.00**	**0.00**

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. L. L. Peterson and B. S. Davie (2003). *Computer Networks – A Systems Approach*. Third edition, Morgan Kaufmann Publishers, San Francisco.

2. C Kaufman, R. Perlman and M. Speciner (2002). *Network Security – Private Communication in a Public World.* Second edition, Prentice Hall PTR, New Jersey.

3. G. G. Xie, J. Zhan, D. A. Maltz, H. Zhang, A. Greenberg, G. Hjalmtysson and J Rexford (2005). On static reachability analysis of IP networks, INFOCOM 2005, 24th Annual Joint Conference of the IEEE Computer and Communications Societies, Proceedings IEEE, Florida, USA, pp. 2170-2183, vol. 3.

4. CNNMoney.com (2006), Can Cisco rediscover its 'growthiness?', http://money.cnn.com/2006/03/10/markets/spotlight/spotlight_cisco/index.htm, last accessed on 25 November 2006.

5. Cisco Systems (2006), Configuring IP Access Lists, http://www.cisco.com/en/US/products/sw/secursw/ps1018/products_tech_note09186a00800a5b9a.shtml#netdiag, last accessed on 25 November 2006.

6. Cisco Systems (2006), IP Services Commands, http://www.cisco.com/en/US/products/sw/iosswrel/ps1831/products_command_reference_chapter09186a00800b3e17.html#wp1017448, last accessed on 25 November 2006.

7. Cisco Systems (2006), Documentation, http://www.cisco.com/public/support/tac/documentation.html, last accessed on 25 November 2006.

8. Insecure.org (2006), Top 100 Network Security Tools, http://sectools.org/, last accessed on 25 November 2006.

9. H. Adiseshu, S. Suri, and G. Parulkar (2000). Detecting and Resolving Packet Filter Conflicts. INFOCOM 2000, 19th Annual Joint Conference of the IEEE Computer and Communications Societies, Proceedings IEEE, Tel-Aviv, Israel, pp. 1203-1212, vol. 3.

10. J. Qian, S. Hinrichs and K Nahrstedt (2001), ACLA: A Framework for Access Control List (ACL) Analysis and Optimization, Proceedings of the IFIP TC6/ TC11 International Conference on Communications and Multi-media Security Issues, Darmstadt, Germany, p. 4.

11. P. Gupta, and N. McKeown (2001), Algorithms for Packet Classification, IEEE Network, vol. 15, issue 2, pp. 24-32.

12, NetBeans (2006), Welcome to NetBeans, http://www.netbeans.org, last accessed on 25 November 2006.

13. IANA.org (2006), Port Numbers, http://www.iana.org/assignments/port-numbers, last accessed on 25 November 2006.

14. IANA.org (2006), Protocol Numbers, http://www.iana.org/assignments/protocol-numbers, last accessed on 25 November 2006.

15. Cisco Systems (2006), Cisco Technical Tips Conventions [Asynchronous Connections], http://www.cisco.com/en/US/tech/tk801/tk36/technologies_tech_note09186a0080121ac5.shtml, last accessed on 25 November 2006.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Geoffrey Xie
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California

4. Professor John Gibson
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California

5. Professor Yeo Tat Soon
   Temasek Defence Systems Institute
   National University of Singapore
   Singapore

6. Ms Tan Lai Poh
   Temasek Defence Systems Institute
   National University of Singapore
   Singapore