ARROS: DISTRIBUTED ADAPTIVE REAL-TIME NETWORK INTRUSION RESPONSE

A thesis presented to

the faculty of

the Russ College of Engineering and Technology of Ohio University

In partial fulfillment

of the requirements for the degree

Master of Science

Karthikeyan Karunanidhi

March 2006

This thesis entitled

ARROS: DISTRIBUTED ADAPTIVE REAL-TIME NETWORK INTRUSION RESPONSE

by

KARTHIKEYAN KARUNANIDHI

has been approved for

the School of Electrical Engineering and Computer Science

and the Russ College of Engineering and Technology by

Shawn D. Ostermann

Associate Professor of Electrical Engineering and Computer Science

Dennis Irwin

Dean, Russ College of Engineering and Technology

KARUNANIDHI, KARTHIKEYAN. M.S. March 2006. Electrical Engineering
ARROS: Distributed Adaptive Real-Time Network Intrusion Response (96 pp.)

Director of Thesis: Shawn D. Ostermann

Research in Intrusion Response has shown that the success rate of an attack increases with time. With attacks becoming sophisticated and automated, the response to these attacks still remains a time-consuming manual process. An *active response* system is a mechanism that can be used in conjunction with an intrusion detection system (IDS) to provide a network administrator with the capability to respond to an attack automatically when it has been detected. Active Real-time RespOnse System (ARROS) is an active, distributed, adaptive, and real-time Intrusion Response System (IRS) that provides Intrusion Response capabilities to INBOUNDS (Integrated Network Based Ohio University Network Detective Service), a network-based, real-time, hierarchical intrusion detection and response system being developed at Ohio University. ARROS consists of distributed autonomous agents that run at various different points on the network it protects. Agents communicate with each other to share information about the network, intrusions, and co-ordinate the response. Each ARROS agent is a fully functional autonomous unit capable of responding to intrusions in a distributed fashion. Coupled with priority queuing for ARROS traffic, distributed response capabilities, and time-bound response the ARROS system is able to provide real-time active Intrusion Response while minimizing adverse effects to the host network.

Approved:

Shawn D. Ostermann

Associate Professor of Electrical Engineering and Computer Science

Acknowledgments

I am forever indebted to my advisor Dr. Shawn Ostermann and Dr. Brett Tjaden for giving me the opportunity to join the INBOUNDS Team. I thank Dr. Ostermann for his guidance, support, encouragement, and patience without which this thesis could not have been completed.

I thank my committee members Dr. Carl Bruggemann and Dr. Jeff Dill for their help. I also thank the members of IRG and the INBOUNDS team for their support.

To mom and dad.

Table of Contents

List of Tables

List of Figures

Chapter 1. Introduction

Networks are getting bigger and faster. The Internet consists of 353 million hosts as of July 2005 [3]. A network that is isolated is safe from intrusions, something that networks in this *information age* cannot dream of. Networks almost always connect to the internet at which point they become vulnerable to intrusions. With networks getting faster and better, the attacks are also now able to spread faster. On January 25th 2003, the Internet community witnessed the attack of the fastest computer worm in history, the Sapphire Worm [16], that doubled in size every 8.5 seconds and infected more than 90 percent of vulnerable hosts on the Internet within 10 minutes.

The number of attacks has increased in the recent years, and has almost doubled every year for the past five years with 137,529 incidents reported in 2003 [8]. Figure 1.1 shows the growth of attacks from year 1988 to year 2003.

Traditionally the job of detecting intrusions is carried out by humans who go through system and event logs, which is a time consuming process that is hardly real-time. By the time the network administrators are aware of the attack, the attack is well underway or in some cases over. Research by Cohen [13] has shown that the probability of success of an attack increases with time delay between the start of the attack and the response. Figure 1.2 shows the increase in the probability of success of attackers with delay in Intrusion Response (IR).

The increase in number of attacks against networks, cost of intrusions, financial loss, and loss of data led to the development of Intrusion Detection Systems (IDSs) that has shifted the job of detecting intrusions from the hands of humans and automated them. Network Intrusion Detection (ID) is the art and science of monitoring networks for activity that may jeopardize the security of the infrastructure under

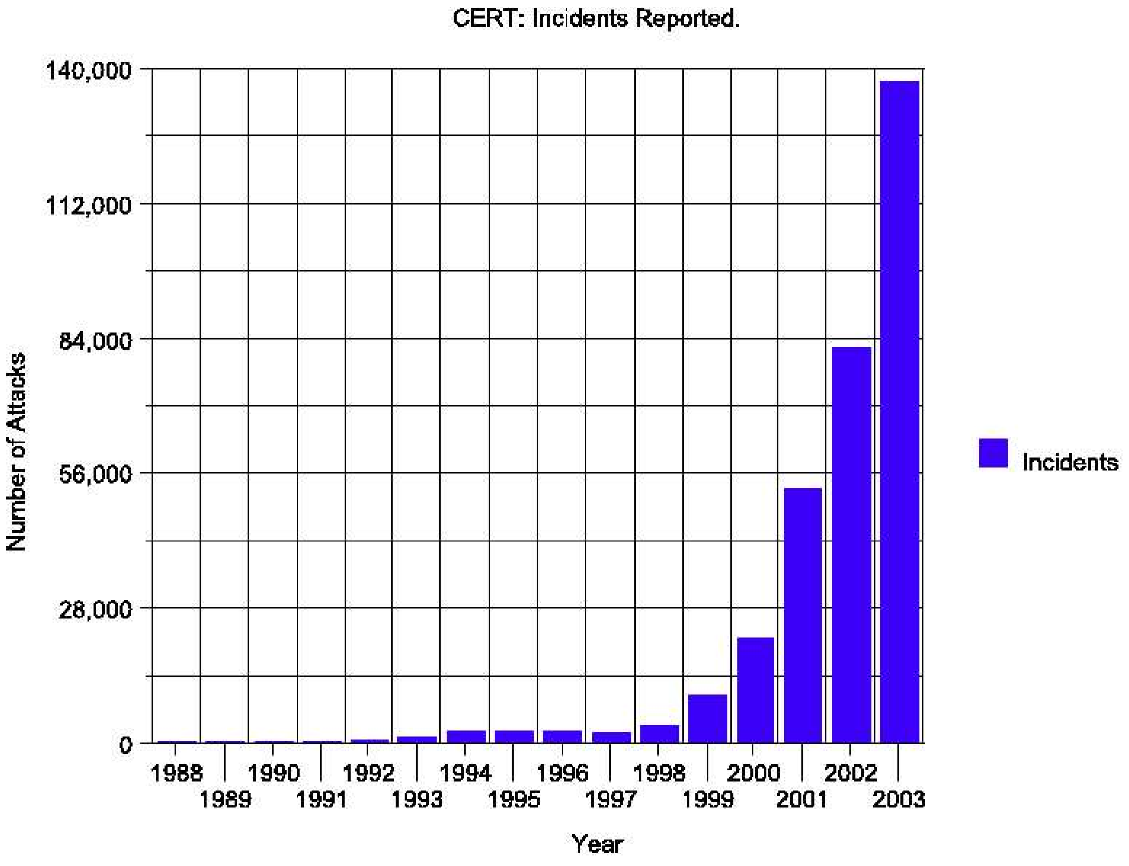


Figure 1.1. CERT: Incidents Reported

surveillance [22]. The advent of IDSs reduced drastically the time it took to detect intrusions, but the job of responding to these intrusions still remained in the hands of the network administrators. The time it took for the administrators to update the firewalls was sufficient enough to cause damage to the network.

An IDS reduces the time it takes to detect intrusions but the dependency on humans for intrusion response forms a bottleneck in the process of IR. The next step in the fight against intrusions is the Intrusion Response System (IRS).

INBOUNDS is an intrusion detection and response system being developed at Ohio University. The INBOUNDS system performs intrusion detection by gathering

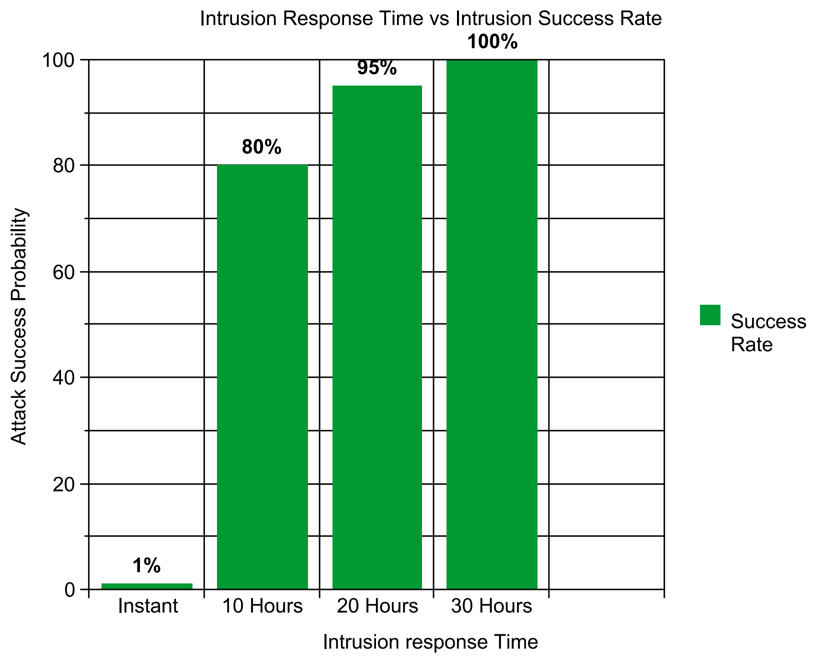


Figure 1.2. Attack Success Rate

network data and analyzing the data for attacks. INBOUNDS detects intrusions using statistical anomaly detection technique [9], Self Organizing Maps(SOM) [20], and Bayesian Networks [26]. Based on our experience with a prototype [11] of an Intrusion Response System (IRS), we have designed ARROS, an Active Real-time Intrusion RespOnse System to provide IR capabilities to INBOUNDS. This thesis describes the ARROS architecture and the extensions to the existing INBOUNDS architecture required to support ARROS and details the working of the ARROS IRS.

1.1 Intrusion Response Systems

An "intrusion" of a resource is defined as "any set of actions that attempt to compromise the integrity, confidentiality, or availability of the resource" [21] and computer intrusion response can be defined as "the process of responding to an intrusion by performing a set of predefined actions". Thus, the goal of an intrusion response system is to respond to intrusions with minimal or no human intervention.

Traditionally the job of responding to intrusion has been in the hands of humans, but with hackers using scripts and software to carry out attacks, it is no longer possible for human response to be fast enough to keep up with the intrusions. This led to increased interest in Intrusion Response systems in the Computer Security community.

An Intrusion Response System (IRS) handles the output from the Intrusion Detection System (IDS). When the IRS receives an alert from the IDS it formulates a response based upon the type of attack. Intrusion Response systems are classified based up the type of response they implement, this classification of IRSs is explained in detail in Sections 2.1.

1.2 Organization of Thesis

We organize this thesis as follows. In Chapter 1 we talk about the current state of Intrusion Response and the need for automated IRSs. In Chapter 2 we provide background information on IRSs, classification of IRSs, related work in the area of IR Research, describe the current INBOUNDS architecture, and provide information on tools used by the ARROS agents to implement intrusion response. In Chapter 3 we describe in detail the enhancements to the INBOUNDS architecture, and the ARROS IRS designed for INBOUNDS. In Chapter 4 we describe our experiences with the prototype ARROS system and the proposed INBOUNDS architecture, and the results. In Chapter 5 we present the advantages and disadvantages of the ARROS system and future work.

Chapter 2. Background

In this chapter we classify the different types of Intrusion Response Systems, discuss the technologies used by the ARROS system to implement intrusion response, related work, and describe in detail the design, architecture, and working of the existing INBOUNDS architecture.

2.1 Classification of Intrusion Response Systems

Intrusion Response is a new and fast developing area in the field of Computer Security. Though research is being carried out in developing these systems there does not exist a formal way of classifying them. In this section we classify them by their response to intrusions. Figure 2.1 shows the classification of Intrusion Response Systems.

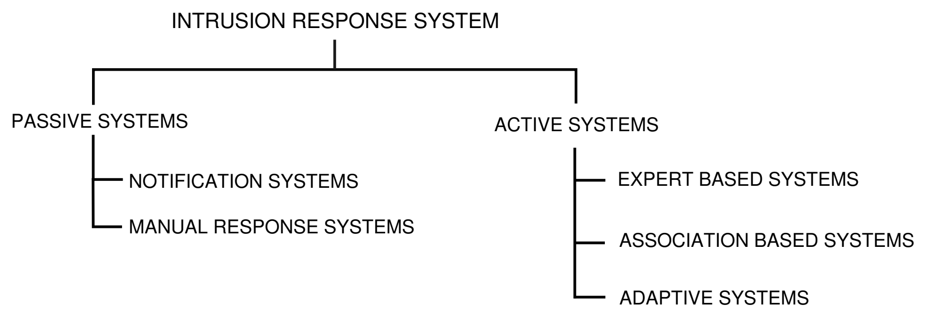


Figure 2.1. Classification of Intrusion Response Systems

Intrusion Response Systems are classified based on their reaction to intrusions as Active Intrusion Response Systems and Passive Intrusion Response Systems. A Passive IRS responds to intrusions either by alerting the system administrator or by depending on the administrator to approve the response. An Active IRS responds to intrusions by modifying the firewall to terminate the connection(s) or limit its bandwidth, with minimal or no human intervention. In the following sections we describe each type of IRS in detail and classify them further into sub-types.

2.1.1 Passive Systems

A passive IRS operates as a notification system. It forwards the details about the intrusion to the administrator by either displaying an alert or sending it as an email or text message to a mobile phone or by providing the administrator with a list of response options to choose from. This allows the administrator to respond to intrusions manually or by applying the predetermined set of rules to modify the firewall. This approach makes the job of the administrator easy by offering ready-made solutions for responding to intrusions. But responding to intrusions still remains a job for the administrator and this provides a time window of opportunity, from the time the intrusion is detected to the time when the first response is launched, for the intruder to carry out the intrusion successfully.

Passive systems are further divided into Notification Systems and Manual Response Systems.

2.1.1.1 Notification Systems

Notification systems alert the network administrator when it receives an intrusion alert from the IDS. The alert can be in the form of an email, text message sent to a cell phone or an audible alert. The job of analysing the intrusion, formulating the response and implementing it on the firewall still remains in the hands of the administrator. The problem with notification type IRS is that the time delay between the intrusion detection and response still exists. The attack might be over and the damage done even before the administrator can get to a computer terminal. The response can take even days if the attack happens over the weekend. Also it takes time for the administrator to analyse the attack, formulate the response and then update the firewall, which is a complex task as the response implementation is different for each type of firewall.

2.1.1.2 Manual Response Systems

Manual Response Systems are a step above the notifications systems in their complexity and use. These systems analyse the alert from the IDS and select the various types of responses, from a list of pre-programmed responses that can be implemented for the current intrusion, and present them to the administrator. This is a step above notification systems and it reduces the time taken by the administrator in analyzing the IDS alert, formulating the response and implementing it on the firewall. But the actual job of responding is still the job of the administrator and the delay from the time of detection to response still exists.

2.1.2 Active Systems

An active IRS responds to detected intrusion by responding with minimal or no human intervention. This dramatically reduces the time window of opportunity for the intruder to carry out his misdeeds. Once an intrusion is detected the IRS modifies the firewall and notifies the administrator about the intrusion and the response that had been implemented. The administrator then has the ability to reverse the response in case of false alarms.

Active or Automatic Response Systems are further classified into Association Based, Expert Based and Adaptive Systems based on how they respond to intrusion and their level of intelligence.

2.1.2.1 Association Based Systems

An Association Based IRS is a simple form of IRS that has a type of response associated with each type of intrusion. Once an intrusion alert is received from the IDS, it looks up the response associated with it and then implements the response in the firewall. The drawback of this type of response is that the list of intrusions and their associated response needs to be kept up to date with every new type of intrusion and it is easy for the intruder to modify the attack so that the IRS is unable to respond.

2.1.2.2 Expert Based Systems

An Expert Based IRS is a much more complex model and has the ability to make decisions based on various parameters attached to the intrusion alert. The type of response can depend on factors such as the severity of the attack, the false-positive rate of the IDS that generated the alert, and the confidence metric of the IDS that the intrusion is real. These systems are harder to fool by adapting to the attack and can respond to unknown intrusion types if other similar intrusions and their responses are known.

2.1.2.3 Adaptive Systems

Adaptive IRSs are intelligent systems that can learn from past successes and failures. They have built-in feedback loops that provide a way for them to learn and correct mistakes. Though these systems are able to learn from past events, the learning is still controlled by humans who provide the feedback on success and failure of the response.

2.2 IPTABLES

IPTABLES [17] is a linux based firewalling system that can perform stateless/stateful packet filtering, network address translation and packet mangling. It is the successor of IPCHAINS [5] and IPFWADM [4] systems. IPTABLES is a generic table structure that defines sets of rules that define/classify match network packets and specify the action to be performed on the matched packet. Several tables of rules exist within IPTABLES framework and each table contains built-in chains and/or user-defined chains. When a packet is received it is matched against each of the rules in the table, sequentially, until a match is found or the end is reached and the corresponding action associated with the chain is performed. The actions that maybe associated with a chain are ACCEPT, DROP, QUEUE, or RETURN.

Figure 2.2 shows the order in which the packets traverse the different chains in IPTables.

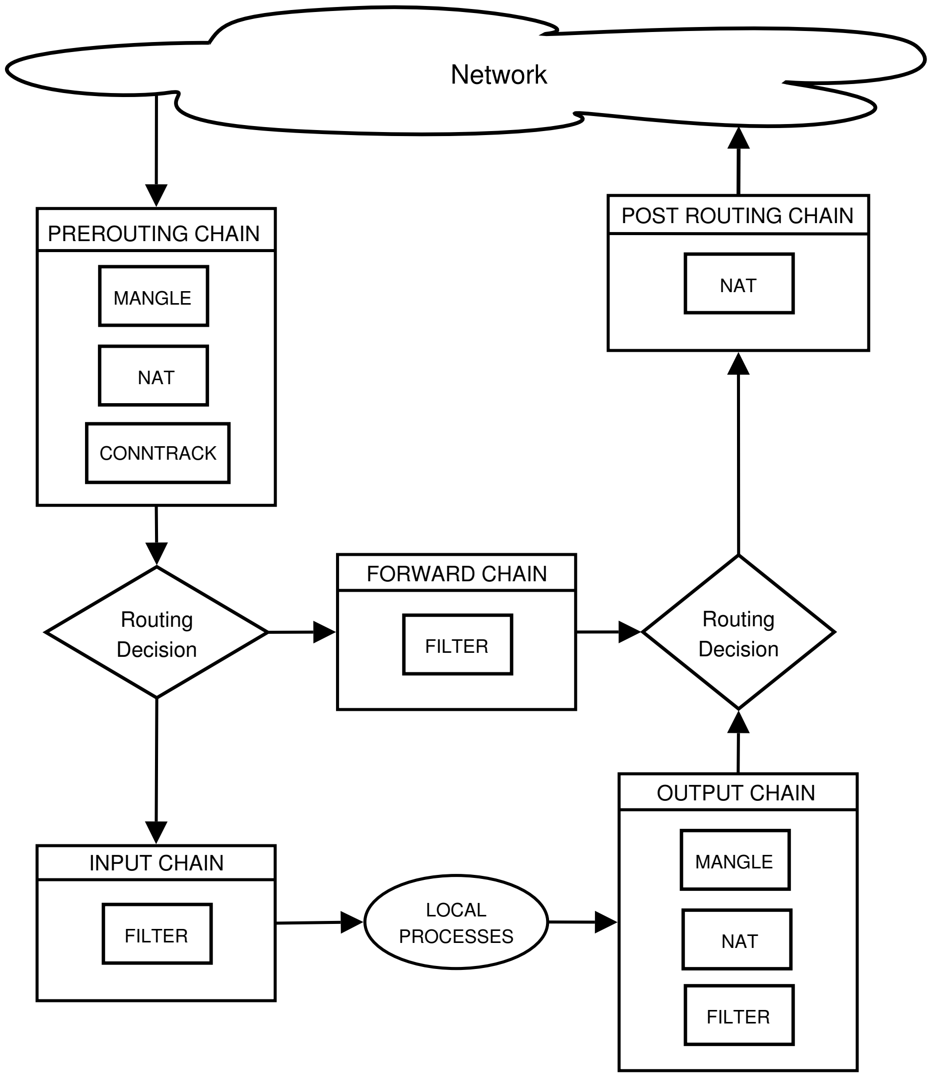


Figure 2.2. IPTABLES

2.2.1 Tables

There are currently three built-in tables - Filter, Nat and Mangle.

2.2.1.1 Filter

The FILTER table is used for filtering packets, the packets are either accepted or dropped by matching them against the rules and by examining their contents. This is the default table that contains the built-in chains INPUT, OUTPUT and FORWARD.

- INPUT - For packets coming into the box.
- FORWARD - For packets that are being forwarded through the box.
- OUTPUT - For packets that are generated by the box, going out.

ARROS agents on linux routers utilize the FORWARD chain in the Filter table to insert rules to respond to intrusion by dropping packets, closing connection, and blocking intrusions.

2.2.1.2 NAT

The NAT table is used to perform Network Address Translation on packets, i.e. to alter the packets source and destination fields. The NAT table is traversed only by packets that creates a new connection, SYN packets and all the consequent packets from that connection have the same operation performed on them. It contains three built-in chains: PREROUTING, OUTPUT and POSTROUTING.

- PREROUTING - For altering packets that come in.
- OUTPUT - For altering packets locally generated by the box.
- POSTROUTING - For altering packets before going out of the box.

2.2.1.3 Mangle

The mangle table is a special form of packet alteration. The mangle table is used to alter values in the packet itself. Some of the values that can be altered by using the mangle table are TOS, TTL, and can also be used to mark the packet with a special value. It contains the PREROUTING, POSTROUTING, INPUT, FORWARD and OUTPUT built-in chains.

- INPUT - For packets coming into the box.
- FORWARD - For packets that are routed through the box.
- OUTPUT - For packets locally generated by the box, before routing.
- PREROUTING - For packets, before routing.
- POSTROUTING - For packets that are about to go out.

ARROS agents on linux routers utilize the PREROUTING, and OUTPUT chains in the MANGLE table to mark ARROS communication traffic so that the communications between ARROS agents are given priority in router queues. The ARROS agents also use the PREROUTING, and OUTPUT chains in the MANGLE table to respond to bandwidth hogging intrusions by marking the packets in the connection to be throttled back.

2.2.2 Targets

Each firewall rule, in the table, specifies the criteria for the packet and also a target. If a packet matches a rule then the fate of that packet is decided by the target specified in that matched rule. The types of targets are ACCEPT, DROP, QUEUE and RETURN.

- ACCEPT - Lets the packet through.
- DROP - Drops the packet on the floor.
- QUEUE - Passes the packet to the userspace if it is supported by the kernel.
- RETURN - Stops the packet from traversing the chain and returns it to the calling chain.

2.3 HTB Traffic Shaping

Hierarchical Token Bucket (HTB) is a hierarchical class-based link-sharing queuing discipline written by Martin Devera to replace the Class Based Queuing (CBQ) Qdisc. HTB is faster than CBQ and is available on a standard Linux kernel beginning from kernel 2.4.20.

Figure 2.3 shows the HTB classes setup used by the ARROS agents to shape the traffic on the network.

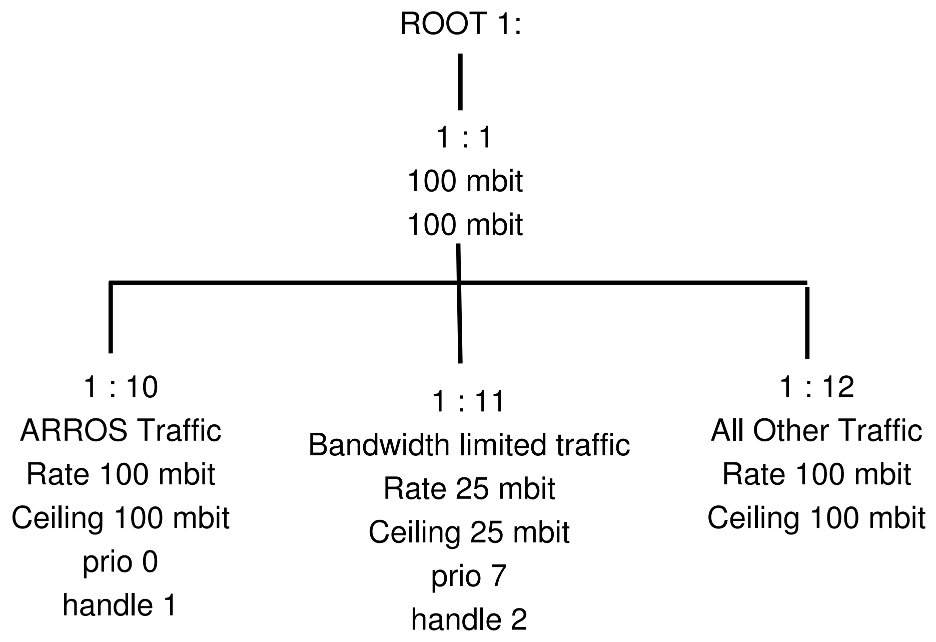


Figure 2.3. HTB Classes

Some definitions,

QDisc - A QDisc, queueing discipline, is scheduler for arranging or rearranging packets leaving the interface. FIFO is an example of a simple scheduling model.

Class - A class can exist only inside a classful QDisc and can contain a number of child classes or a single QDisc.

Filter - Filters contain classifiers that are used to classify/select packets based on certain characteristics and direct the selected packets to a sub-class. Filters can be attached to a classful QDisc or a class.

Classful Qdisc - A class based QDisc can contain classes to which filters can be attached.

Token Bucket Filter (TBF) is a queueing discipline that uses tokens and buckets to shape the traffic on an interface. Each bucket can hold a specified number of tokens and the bucket is replenished at a given rate. A packet is transmitted only if a token is available.

HTB consists of a hierarchy of buckets that uses tokens and borrowing models to provide granular, yet sophisticated ways for controlling the traffic through an interface. The traffic is shaped in the leaf classes and the inner classes in the hierarchy and specify the borrowing model. Each class has various parameters attached to it:

- rate - the minimum desired speed, similar to the committed information
- rate (CIR) or the guaranteed bandwidth.
- ceil - the maximum speed allowed
- burst - HTB can dequeue burst bytes before the arrival of more tokens
- quantum - the parameter used to control borrowing
- prio - priority, between 0 and 7, of a leaf class.

2.4 Access Control Lists

CISCO routers implement filtering using Access Control Lists(ACLs) [2]. An Access Control List is a list of statements, where each statement defines a particular pattern of IP packets and has a permit or deny rule associated with it, and has to be associated to a particular interface for Ingress or Egress for filtering. For every packet that enters the router, the list is scanned top to bottom, in the order that it was entered, and when a match is found then the packet is permitted or dropped based on the rule in the matching ACL entry.

2.5 Related Work

The concept of Intrusion Response has steadily gained popularity in the field of Computer Security over the past decade. Past and ongoing research in this area has led to the development of many interesting approaches with as many as 56 systems [15] till date. In this section we examine four systems that have goals similar to ARROS.

2.5.1 Cooperating Security Managers

Cooperating Security Managers (CSM) [25] is a host based Intrusion Detection and Response System that cooperates with other CSMs in a distributed environment, without a central controller. CSM takes a proactive approach to intrusion response. It uses Fisch DC & A taxonomy to classify attacks and response. When an intrusion is detected, it assigns it a level of suspicion to it and the response is chosen based on the level assigned to the intrusion. There are eight different response sets used by the CSM and each response set has one to fourteen different response actions. However the CSMs do not remember and learn from false positives and negatives, do not have a measure of the success or failure of the chosen response and do not differentiate between new and old attacks.

2.5.2 EMERALD

Event Monitoring Enabling Responses to Anomalous Live Disturbances (EMERALD) [19] is a distributed misuse and anomaly IDS coupled with a real-time Intrusion Response system that has been developed for use in a large networks with variety of components. The architecture comprises of a hierarchy of monitors that receive reports from the analysis components and invokes the corresponding response using the response handlers. The type of response is based on two metrics, the threshold and the severity metric. The threshold metric indicates the certainty that the intrusion is real and not a false positive and the severity metric indicates how severe a response is required. However the EMERALD monitors do not remember and learn from false positives and negatives, do not have a measure on the success or failure of the chosen response and do not differentiate between new and old attacks.

2.5.3 AAIRS

Adaptive Agent based Intrusion Response System (AAIRS) [7] is host based Intrusion Detection and Response System that employs several metrics to determine if the intrusion is new or an old attack and also remembers and learns from false positives and the success or failure of a response. The host being protected is monitored by multiple IDSs. The alerts from the IDSs are converted to a common format by the Interface Agent that also keeps track of the false positive and negative rate of each individual IDS. The Interface agent then sends the alert to the Master Analysis agent along with an attack confidence metric. The Master Analysis agent maintains an event history list and classifies each intrusion as new or existing attack. If the attack is an old one the Master Analysis agent send it to the Analysis agent that is handling the attack, else if it is a new attack then it creates a new Analysis agent that formulates a response plan. The Analysis agents use the Response Taxonomy agent to classify attacks, the Policy Specification agent to formulate the response, and the Response Toolkit to implement the response. However the AAIRS is a host based system and cannot provide distributed response. Though the system uses an attack confidence metric that is generated by the Interface Agent, the actual IDS confidence metric is updated by humans and is not automated.

2.5.4 INDRA

Intrusion Detection and Response Architecture (INDRA) [14] takes a distributed, P2P approach to intrusion response and can respond proactively or retroactively to intrusions. When an intrusion is detected by a security daemon it multicasts the intrusion information to other daemons in the network that implement response based on that information. The architecture consists of trusted peers that share rumors, facts and trusted information to guard the network. The INDRA nodes use publish-subscribe multicast mechanism or rumor spreading models to communicate among themselves. The architecture consists of Watches that look for any suspicious activity on the host or the network and generate alerts. The Listeners combine all the alerts from the watchers and send alerts to the Access Controllers that restricts the users access to the computer that generated the alert. Reporters are used to communicate with other INDRA nodes. The INDRA nodes have a plug-in feature that allows new functionalities to be added at runtime.

2.6 INBOUNDS

The INBOUNDS system is designed to detect attacks early and respond at an appropriate points in the network that prevents further attack propagation. The goals of the current INBOUNDS are [9]: Run continually, Be fault tolerant, Resist subversion, Be scalable, Operate with minimal overhead, Be easily configurable, Cope with changing system behavior, Be difficult to fool with, and Detect never before seen attacks.

Figure 2.4 shows the existing INBOUNDS architecture diagram. The INBOUNDS project is currently under development with some of the modules in their early stages of development. The goal of this section is to present a high-level view of the INBOUNDS system so as to give proper context for the description of the ARROS System.

The heart of the INBOUNDS system is the Intrusion Detection Engine. This engine makes the decision on whether the network connection being analyzed looks normal or if it is anomalous and should be classified as an intrusion.

2.6.1 Existing Architecture

The existing INBOUNDS system gathers raw network data using tools such as tcpdump [23] and ethereal [1]. It uses a program named TCPurify [10] that truncates the raw packets by removing the data payload beyond the standard IP and TCP/UDP headers, providing privacy. The ID module detects intrusions using Statistical Anomaly Detection [9] method and Self Organizing Maps [20]. The current

INBOUNDS system implementation can detect intrusions/anomalies in real-time and respond to the intrusions locally [11].

The current INBOUNDS has successfully implemented the ability to run continually, be scalable, operate with minimal overhead, easily configurable, and detect never before seen attacks, from it list of goals. The current INBOUNDS system has two IDS modules, the SOM module and the NAID module, that are able to detect intrusions such as Buffer Overflow attacks, HTTP Tunnels, Mail Bombs and P2P file sharing services. The SOM IDS module uses Self Organizing Maps to detect never before seen intrusions and has a false positive rate of 1.4%. It is an offline IDS that can be easily adapted to be a real-time IDS. The NAID module is a real-time IDS module that uses a Statistical approach to perform both host and network-based intrusion detection.

Figure 2.4 shows the current INBOUNDS architecture diagram, is based upon the earlier INBOUNDS architecture [20]. The goal of this section is to provide a description of the various modules in the existing INBOUNDS system.

2.6.1.1 Data source

The Data Source module provides network data packets as input to the Intrusion Detection Engine. Each network monitored by INBOUNDS requires a data source module. The data source captures raw network data, which can typically be done using tools such as tcpdump [23] and ethereal [1]. However, INBOUNDS uses a program called TCPurify [10], a packet sniffer program that strips the data payload, masks the source and destination IP addresses in the IP header and provides only the needed headers for anomaly detection.

2.6.1.2 Data Processor

The data processor module obtains network data from the data source and processes it. The code that performs the processing is present in the INBOUNDS module [12] for TCPtrace [18]. TCPtrace is a TCP analysis tool that reads network data captured from a data source and groups the packets into conversations, and analyzes the

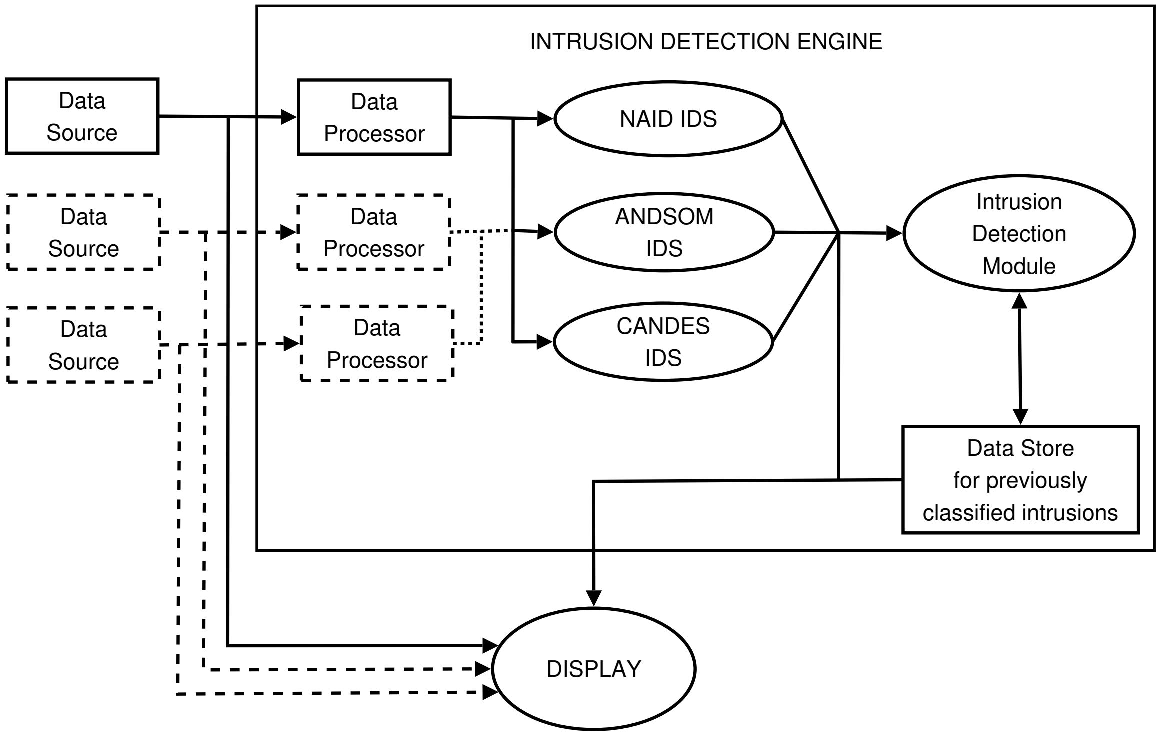


Figure 2.4. Existing INBOUNDS Architecture

individual conversations. The output from the INBOUNDS module are messages that are used by the intrusion detection module of INBOUNDS. Current IDS modules of INBOUNDS use three kinds of messages named O, U and C messages. These messages are reported for each connection captured by the data source.

2.6.1.3 Display

The Display module is used to give a real-time display of the connections in and out of the network being monitored by INBOUNDS. The program *networkgraphserver* has been written for this purpose. This program is written in Java and gives a real-time picture of the network, with each host in the network being represented by icons and the connections between hosts indicated by lines between hosts. This module is a work in progress.

2.6.2 ANDSOM Module

The Anomalous Network-traffic Detection with Self-Organizing Maps (ANDSOM) [20] module uses the Self-Organizing Map algorithm to build profiles of normal network traffic. The profiles built are later used to make a decision on whether a network connection is normal or anomalous.

2.6.3 NAID Module

The Network Anomaly Intrusion Detection (NAID) module [9] detects anomalies in real-time. It builds a statistical profile of the services during normal operation and detects anomalies by comparing the real-time metrics against the stored profiles. The NAID module uses the following network metrics to build the profile of the services:

1. Interactivity: number of questions during a time interval normalized to 1 second
2. ASOQ: average size of questions during the time interval in bytes
3. ASOA: average size of answers during the time interval in bytes.
4. QAIT: sum of idle question-answer time during the time interval normalized to 1 second
5. AQIT: sum of idle answer-question time during the time interval normalized to 1 second
6. NOC: number of connections in the network

The module observes the network metrics on a network level which enables it to detect abnormalities on a service level such as an email macro virus, which would be undetectable on a single-host/per-connection level, and also observes the network metrics on a host level which enables it to detect attacks on a host level such as a buffer overflow attack, in which the finger daemon exhibits a high level of interactivity.

2.6.4 Intrusion Detection Module

The INBOUNDS system has multiple IDS modules that use different techniques to detect intrusions. The Intrusion Detection Module receives alerts from the various IDS modules and corroborate the alerts and generate an IDS alert based on alerts from the past, and the confidence metric of each IDS module. This module is a work in progress.

Chapter 3. New INBOUNDS Architecture

In the fast networks of today, where it takes only a matter of minutes for an attack to infect all available hosts in the internet, there is a need for a system that will be able to respond to such an attack as it happens.

Previously the task of ID was performed by humans who went through mountains of raw data and network logs to detect intrusions. But the need for detecting intrusions in real-time led to the development of IDSs that took the task of ID from humans and automated them. There is now a similar need in the area of IR where it is no longer possible for the administrator to respond to intrusion fast enough to be effective in real-time.

One of the important goals of the new INBOUNDS system is autonomous active response to intrusion events. This goal requires enhancements to other areas of the INBOUNDS system such as distributed detection and attack correlation, tracking the path of an intrusion through the network, and response at a node appropriate to the type and severity of attack. These additional features are described briefly in the next section followed by the ARROS intrusion response system in Section 3.2.

3.1 Enhancements

Figure 3.1 shows the new architecture of the INBOUNDS system. This architecture is a work in progress with the various modules in their early stages of development. We have developed a prototype for the ARROS system and its communication protocol and conducted experiments to prove the feasibility of the new INBOUNDS architecture [11]. In this section we provide a description of the enhancements made to the INBOUNDS architecture and set the context for the description of the ARROS IRS.

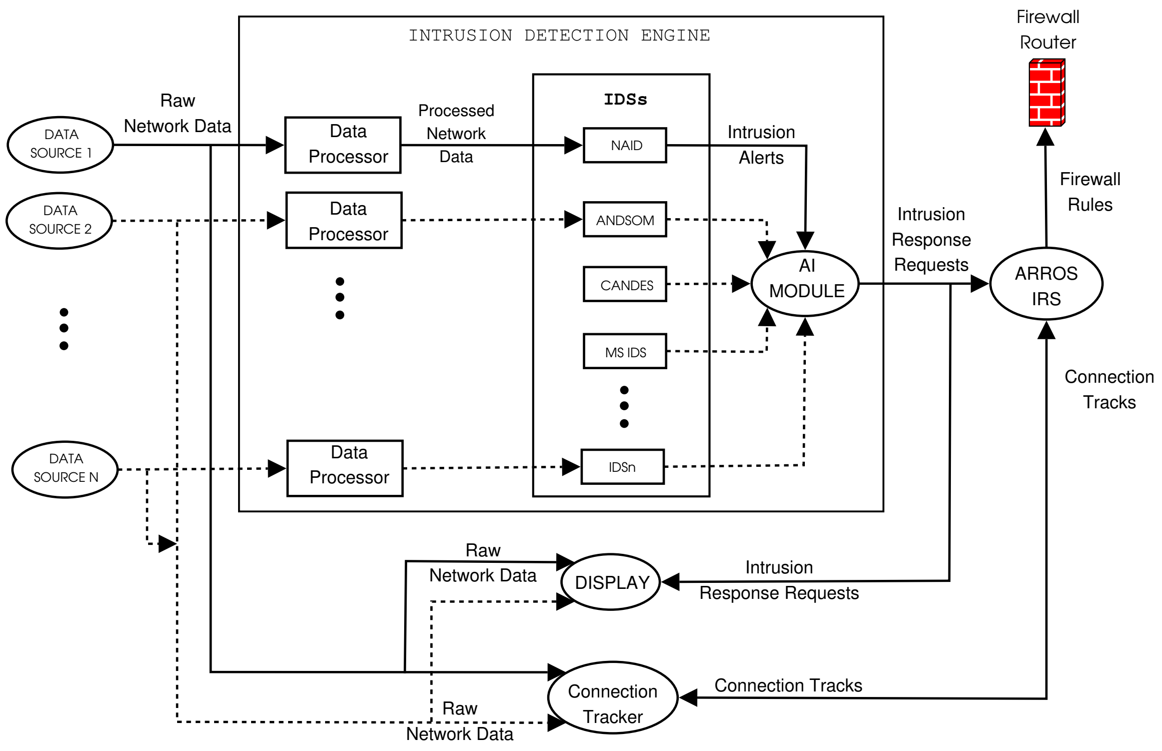


Figure 3.1. New INBOUNDS Architecture

3.1.1 IPTraceback Marker Module

The ability to respond at a location other than the location where the intrusion is detected is possible only if the path of the intrusion through the host network is known. The path of the intrusion through the host network is traced and is made available to ARROS agents by the IPTracker module. This module consists of two sub-modules, the IPTraceback Marker and the IPTraceback Tracker. The IPTraceback (IPT) Marker module runs on the border gateways of the host network and grabs packets as they enter the network. It modifies the packets using the mangle table of the Linux IPTABLES [17] and turns on the IP Record Route option for SYN, SYN+ACK and all UDP packets. Figure 3.2 shows the IPTraceback Marker module.

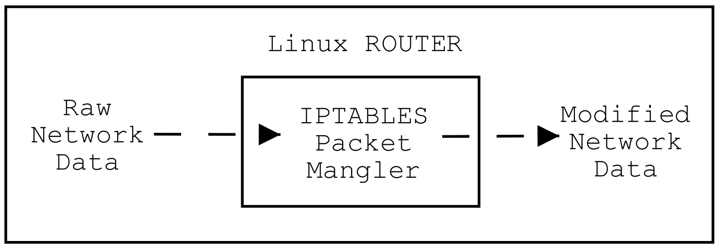


Figure 3.2. IPTraceback Marker Module

3.1.2 CANDES IDS Module

CANDES [26], Causal Tree Anomaly-based Network-Based Detection System, is a new real-time IDS module that uses Bayesian Networks for intrusion detection. The CANDES module receives information about the connections from the data processor module. It calculates the confidence interval for each of the connection parameters, and sets the range for classifying a value as no anomaly, low anomaly, medium anomaly, and high anomaly. During runtime the connection parameters are calculated and compared against the values obtained during the learning phase, and based on this comparison the connection is then assigned one of four values: no anomaly, low anomaly, medium anomaly, and high anomaly.

3.1.3 IPTraceback Tracker Module

The IPTraceback Tracker module runs on the gateway router of each subnet or at a location where it has access to all the traffic in the subnet. Alternatively it can receive network data packets from the Data Source module. The IPTraceback (IPT) Tracker module grabs the network data packets and looks for packets destined to machines in the monitored network and have IP Record-route option turned on. It builds a database that contains the record route information for all active/open connections. The record route information for a connection is deleted from the database after a definite amount of time (this time is tunable) after the connection has been closed. Figure 3.3 shows the IPTraceback Tracker module.

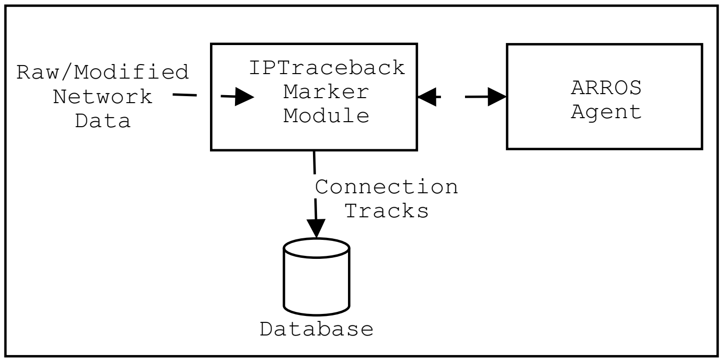


Figure 3.3. IPTraceback Tracker Module

3.1.4 Decision and Intelligence Module

The Decision and Intelligence module is in part the Intrusion Detection Module in the previous INBOUNDS architecture. This module is a work in progress that will be designed to perform the following functions:

- Recieve and correlate intrusion alerts from multiple IDSs.
- Calculate the intrusion confidence metric and response severity metric.
- Formulate intrusion response plans.
- Provide an interface for administrators to review and rollback/modify responses.
- Assign confidence metric to IDSs based on past success and false positives/negatives.
- Update IDS confidence metric for each IDS based on the feedback from the administrator.
- Record intrusion alerts, and the success/failure of the response formulated.
- Learn from the response history to better formulate response plans.

The Decision and Intelligence module consists of the Intelligence Module (IM) and the Decision Module (DM). The IM receives intrusion alerts from one or more IDSs and gathers any intelligence that can be construed from the alerts of various IDSs and past intrusion events. The information acquired is used by the DM to reach a decision on the type of response. The decision is then sent as a request to the ARROS agent, which implements the response. Figure 3.4 shows the IM and the DM.

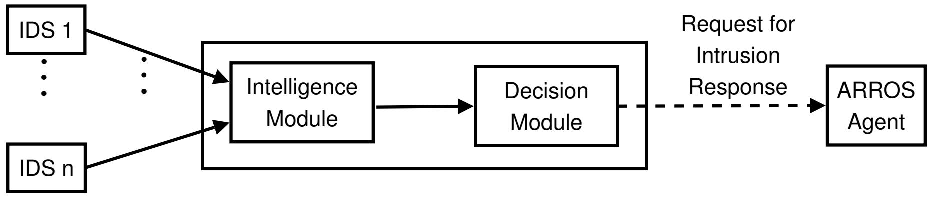


Figure 3.4. Decision and Intelligence Module

3.1.5 Intrusion Response Module

On of the major enhancements to the INBOUNDS architecture is the addition of an automated intrusion response module. The ARROS intrusion response system provides IR capabilities to INBOUNDS. The goal of the Intrusion Response module is to take active response on the connections being perceived as intrusions by INBOUNDS. Some of the response actions against intrusions include: active firewall blocking of a specific network connection, reduction of priority for that connection or that class of network connections at the border router, to name a few. The ARROS intrusion response system is described in Section 3.2.

3.2 ARROS Intrusion Response System for INBOUNDS

The ARROS system provides intrusion response capabilities to INBOUNDS. The ARROS system architecture is a distributed agent-based architecture that consists of a number of autonomous ARROS agents running on various different subnets in the host network that they protect. The ARROS agents communicate with each other to share intrusion data and co-ordinate the response. ARROS agents receive requests to respond to intrusions from the DM and other ARROS agents in the network.

When an ARROS agent receives a request, it inserts a firewall modifying rule that blocks/bandwidth-limits the intrusion.

ARROS takes a distributed active approach towards intrusion response. One of the important goals of the new Inbounds architecture and ARROS is to provide faster response to the intrusions than what is manually possible. The new architecture is designed to be distributed, wherein every node has the same capability, intelligence and can perform independent if the need arises. This ensures that there is no single/central point of failure/vulnerability. The communication between the agents is kept to a minimum, while not compromising performance, so that the system itself does not load the network or gives the attackers a chance to use it as a means for mounting a DOS attack.

Figure 3.5 shows the architecture of the ARROS IRS.

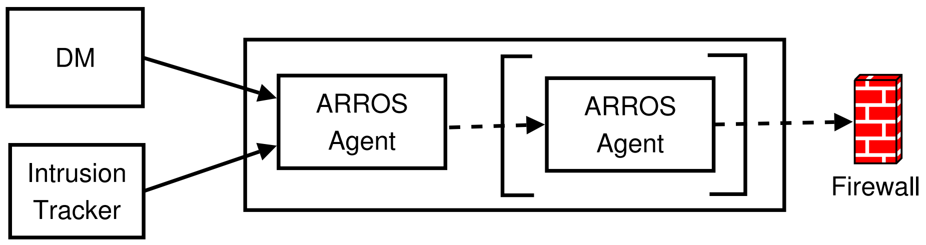


Figure 3.5. Intrusion Response System

3.2.1 ARROS System Goals

ARROS is designed to provide intelligent, distributed, and real-time intrusion response capabilities to INBOUNDS.

The goals of the ARROS system are

- Real-time response.
- Scalability.
- Ability to rollback responses automatically and manually.

- Intelligent goal-oriented response.
- Graceful degradation.

The ARROS architecture has been designed to accommodate all of the above goals. The current ARROS architecture and the prototype ARROS implementation is designed to work on a LAN with a few tens of ARROS agents, and isn't intended to scale well with a large increase in the number of agents; this is an issue to be addressed in future implementations using flooding algorithms to improve scalability. The agents are fully functional autonomous units that are fully capable of responding to intrusions without depending on other agents or applications. Each agent is designed to interact with other agents in the network to share information and coordinate response to respond to the intrusion in the best possible way. But if the agent is not able to communicate with the other agent, it is still able to respond to the intrusion.

ARROS agents support the following types of responses:

- *Bandwidth limit connections*
  - Locally
  - At another point in the network
  - At all points along the path of the intrusion through the network
  - At all points in the network
- *Block limit connections*
  - Locally
  - At another point in the network
  - At all points along the path of the intrusion through the network
  - At all points in the network

3.2.2 ARROS System Architecture

Research in Automated Intrusion Response [24] has shown that for automated response to be successful it needs to satisfy certain criteria. The automated response system should be flexible, dynamic, efficient, easy to use, and minimize negative effects on the host network. These criteria greatly influence the design of the ARROS IRS, and has the following features:

- Autonomous
- De-centralized
- Be scalable
- Degrade gracefully
- Be fault tolerant
- Adaptable response

The agents consist of five modules, based on their functionality. We provide a short description of the function of each module and their interactions with each other. The modules are described in detail in the next section.

- Response Engine - Listens for intrusion response requests from the Decision and Intelligence Module, and from other ARROS agents.
- Data Manager - Maintains the information about ARROS agents in the network, and about the responses that are currently implemented in the firewall.
- Timer Module - Tracks the lifetime of each intrusion response.
- Rule Builder - Formulates firewall modifying commands.
- Communication Module - Handles inter-agent communication and communication with other modules.

Figure 3.6 shows the ARROS system modules. When the Decision and Intelligence module receives an intrusion alert from the IDS, it corroborates the alert with alerts from other IDSs, past IDS alerts, and the confidence metric of each of the IDS, and generates a response plan for the intrusion. It then sends the intrusion response plan to the local ARROS agent as an Intrusion Response Request (IRR). The Response Server receives the IRR and based on the type of response, it passes the IRR to the Communication Module to be sent to other ARROS agents or to the Data Manager. The Data Manager writes the IRR into the database, updates the Timer Module with the expiration time of the IRR, and then forwards the IRR to the Rule Builder. The Rule Builder formulates the firewall modifying command and executes it on the firewall (control point). When the rule expires the Timer Module alerts the Data Manager and the Rule Builder to delete the intrusion response from the database and the firewall respectively. If the Rule Builder fails to modify the firewall it uses the Communication Module to send the IRR to other ARROS agents in the network, until it finds an agent that is able to respond successfully.

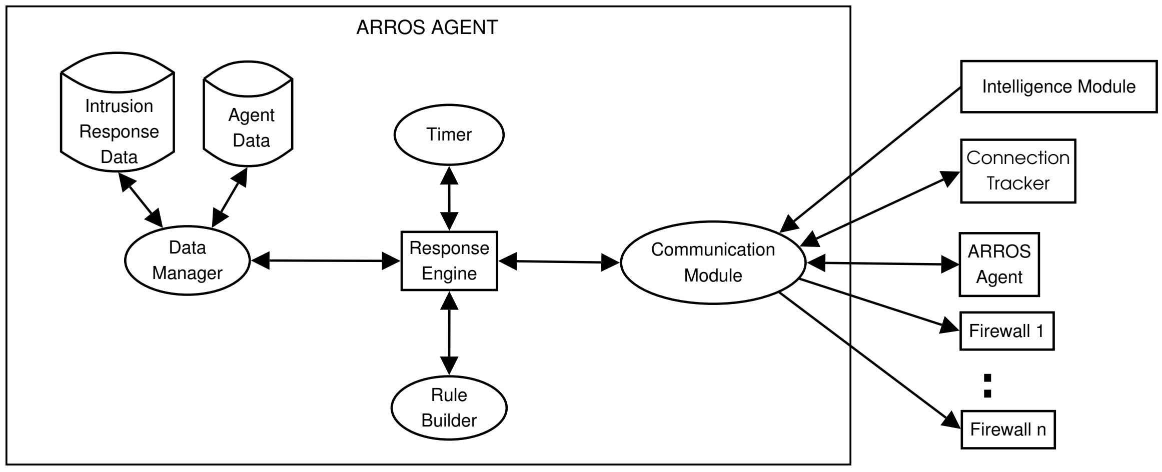


Figure 3.6. ARROS Data Flow Diagram

3.2.3 ARROS System Modules

The ARROS agent is designed to work as an independent entity and to make use of other distributed ARROS agents when possible, to provide the best type of response. The agent is made up of five modules. In this section we describe in detail the working of each module.

3.2.3.1 ARROS Communication Module

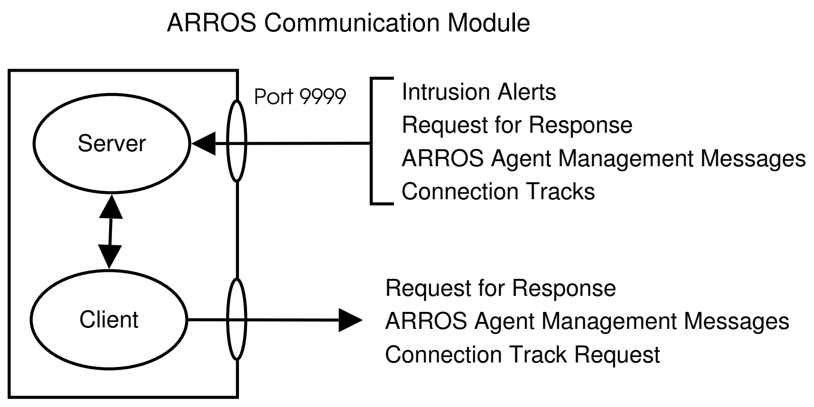


Figure 3.7. ARROS Communication Module

Figure 3.7 shows the ARROS communication module. The communication module handles all the inter-agent and intra-agent communications of the agents. It consists of two sub-modules, the Server and the Client. The server runs on port 9999 and handles all the incoming connections from the IDS and other ARROS agents. The client module handles connections to the IPTracker and other ARROS agents.

The communication module uses TCP connections for all its communication purposes. Using UDP as the communication protocol, though certainly reducing the overheads like three-way handshake, does not provide the reliability required. Using TCP provides the communication module with the benefit of reliable best effort connections.

Trust is a very important issue in designing a distributed and automated IRS. The architecture of the ARROS system and the prototype implementation of the

Communication Module do not address the issue of *trust*. All inter-agent and intra-agent communications in the prototype ARROS implementation is un-encrypted and implicitly assumes that all incoming communications are from trusted sources. The issue of trust and encryption in communications involving distributed agents is beyond the scope of this research and many turnkey solutions exist that address this issue. Some of the possible solutions that can be easily integrated into the ARROS architecture are:

- Using public and private keys to encrypt all communications.
- Configuring the agents to only accept TCP connections from IP address of agents that it knows from its agent table entries.

A typical communication exchange involves the sending of a header that tells the receiver the type and size of the data to follow, followed by the data itself. The process of communication between the ARROS agents is shown in Figure 3.8. The smallest communication initiated by an ARROS agent consists of steps 1 and 2, with optional steps 3 and 4.

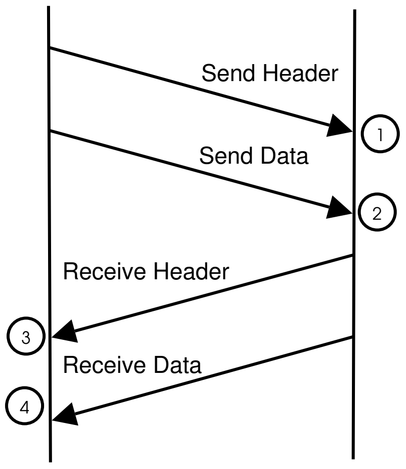


Figure 3.8. Steps in a Typical ARROS Agent Communication

The header message format is shown in figure 3.9. The *type*, *size*, and *number* fields indicate the type of the communication/data-transfer being initiated, the size of the data, and the number of units of data that follow the header.

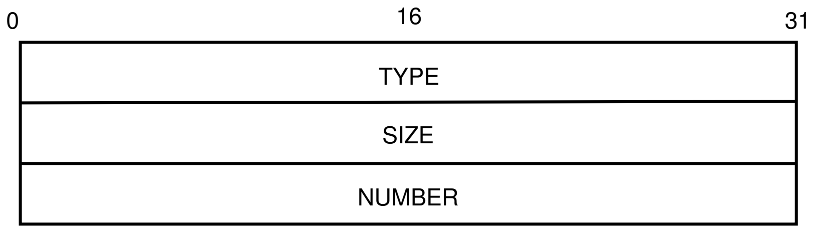


Figure 3.9. Header Message Format

There are eight different types of communication that take place in an operational ARROS agent as shown in Figure 3.10.

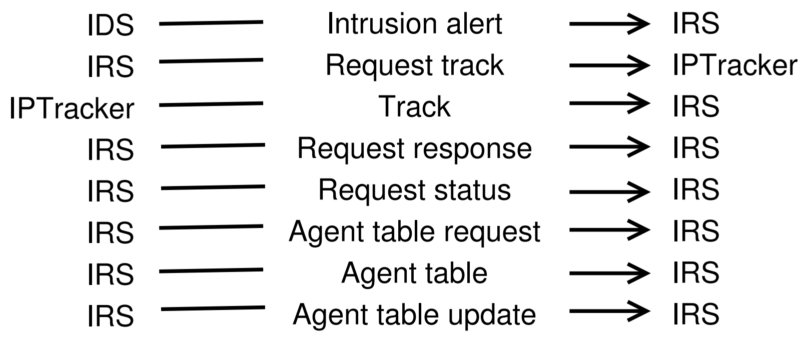

IDS — Intrusion alert → IRS
IRS — Request track → IPTracker
IPTracker — Track → IRS
IRS — Request response → IRS
IRS — Request status → IRS
IRS — Agent table request → IRS
IRS — Agent table → IRS
IRS — Agent table update → IRS

Figure 3.10. Message Types

3.2.3.2 Timer Module

The ability of an automated intrusion response system to rollback responses, either automatically or by human intervention, is important in mitigating loss in the network. The timer module keeps track of the expiration times of the intrusion responses that are implemented by the agent. With each new response the Timer updates the database. When a response expires the Timer deletes the response from the database and from the firewall.

3.2.3.3 Data Manager Module

The Data Manager module writes all responses received by the agent into the database and also keeps track of all the other agents in the network. There are two data stores that are managed by the data manager module, the rule's file to save response information and the agent table file to save the agent's information.

When the agent receives a new intrusion response request or a new agent table entry it inserts it into its data structure and also writes it into the date store. Also when the agent is shutdown the data manager saves all the intrusion information and the agent table entries into the data store before stopping, ensuring that the information is not lost.

When the agent is started, the data manager retrieves the intrusion responses, that have not expired at that time and sends them to the Response Engine to be reinserted into the firewalls. It also reads the agent table entries from the data store and rebuilds the agent table for use by the agent to communicate with other agents in the network. This ensures that the responses are not forgotten due to any conditions that causes the agent to stop, and also provides the agent with a list of all agents in the network from its last runtime. Figure 3.11 shows the data manager module.

3.2.3.4 Rule Builder Module

The Rule Builder module receives information about the intrusion and the type of response to be implemented from the Data Manager module and formulates the command for modifying the firewall and implements the command on the firewall.

In case of a linux firewall router, running IPTABLES, the rule builder formulates an IPTABLES command to insert a rule into the FORWARD chain in the IPTABLES firewall. The prototype ARROS IRS has the ability to kill or limit the throughput of TCP connections.

If the firewall is a CISCO router, the rule builder formulates an ACL (Access

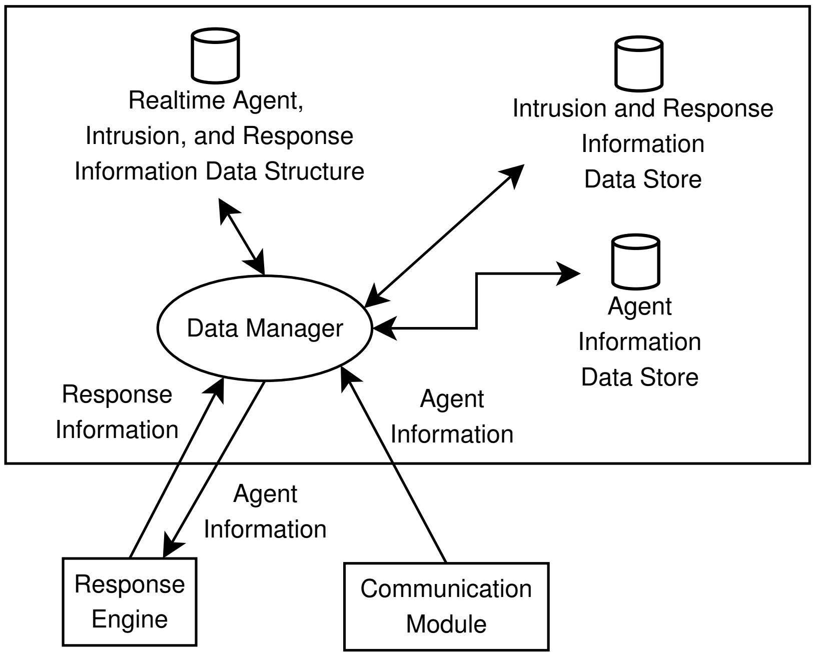

Realtime Agent, Intrusion, and Response Information Data Structure
Intrusion and Response Information Data Store
Agent Information Data Store
Data Manager
Response Information
Agent Information
Agent Information
Response Engine
Communication Module

Figure 3.11. Data Manager Module

Control List) entry to be applied to the corresponding interface on the router and then connects to the router and executes the command in the router. Figure 3.12 shows the Rule Builder module.

3.2.3.5 Response Engine

The Response Engine is the core of the ARROS agent that ties together the various module in the ARROS agent. Figure 3.13 shows the Response Engine. The Response engine is responsible for correlating the path of the intrusion received from the IPTracker and the list of known ARROS agents in the network and their capabilities to choose the best suited response point. If there is a failure in the response the Response Engine chooses the next best response point and works through the list of all possible response points until it finds an agent that is successful in responding to the intrusion.

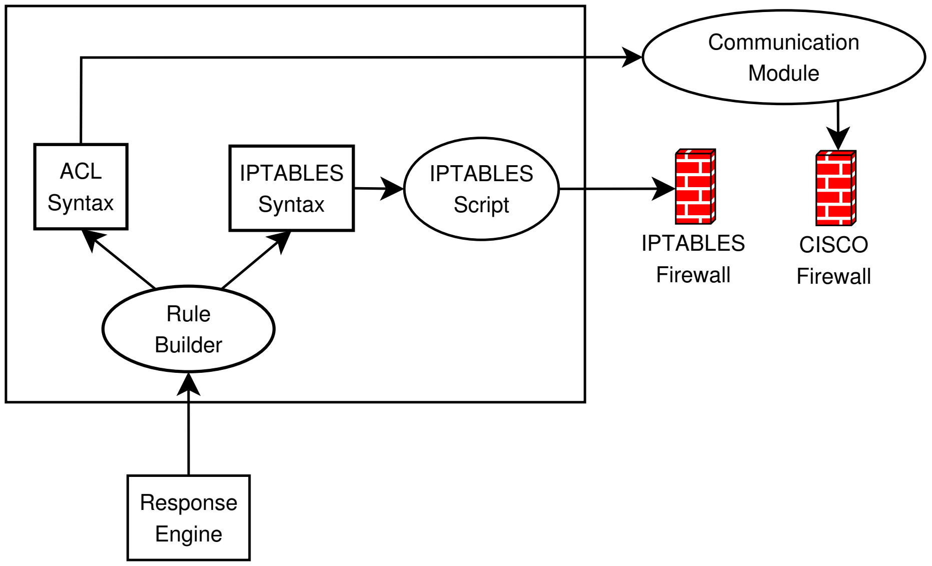


Figure 3.12. Rule Builder

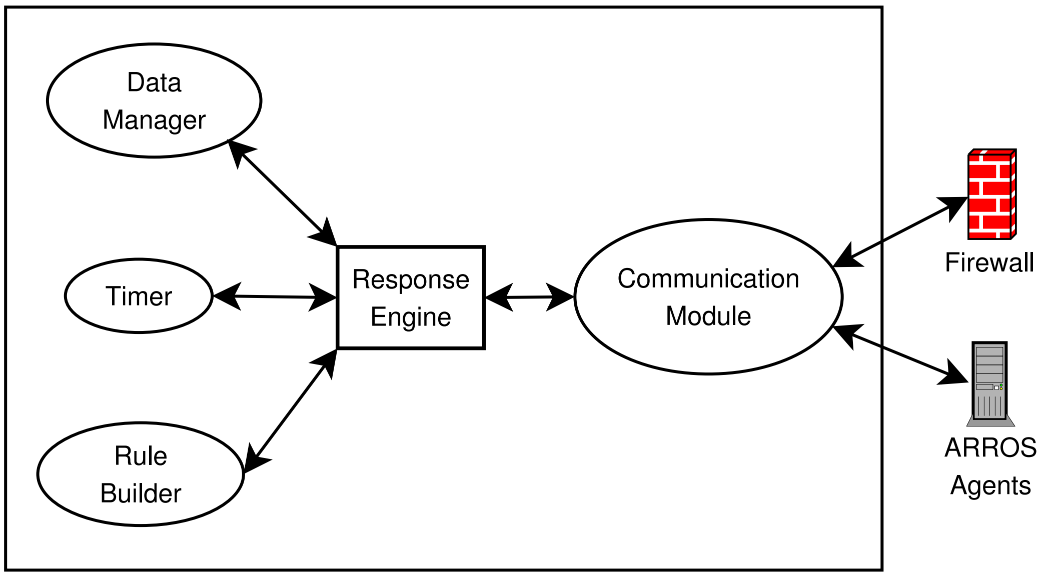


Figure 3.13. Response Engine

3.2.4 ARROS Communication Protocol

The ARROS agents run at various different points in the host network and communicate amongst themselves to share information about the network, about the intrusions, and co-ordinate the response to intrusion to ensure that the response has minimal adverse impact on the host network. The agents communicate using a simple communication protocol that involves exchange of their agent tables, similar to routers exchanging routing tables. The agents share information about themselves by sharing their Agent Table entries. Each ARROS agent has an entry in the agent table with the following information:

- agent identifier
- agent IP
- agent's subnet IP and mask
- list of firewalls the agent controls and their type and capabilities
- extended control (boolean)
- if extended control == true
  - list of firewall IP, firewall type, network IP and mask, firewall capabilities

When a new agent is started in the network it connects to its neighbouring agent and requests the Agent Table entries. When it receives the agent table entries it contacts all agents in the list and shares its information with them.

When an agent sends a request for intrusion response to another agent in the network, it sends the intrusion information to the other agent. The receiving agent processes the request and responds with a success/failure message.

In both types of communication the actual data is preceded by a standard header 3.9 that informs the recipient about the type and size of the data to follow.

3.2.5 ARROS Agent Configuration

Each ARROS agent has the capability to operate autonomously when it is alone in the network. But when the agent runs in a network along with other distributed agents, the agents exchange information about intrusions and their subnet. The agents work together to coordinate the response to intrusions. When a new agent, say agent A, is started, it is provided with the id and the IP of one other agent, its neighbour, say agent B, that is already a part of the ARROS agent network. Agent B provides information about the other agents that are in the network. Agent A then connects to agent B and other agents in the list and exchange information about the network. When there is a change in an agents agent-table, the updated information is shared with all the other agents. This ensures that within a short period of time all agents learn about any changes to the agent network without using periodic updates via multi-cast or broadcast which is an unnecessary overhead.

The use of a *neighbour* agent to assist a new agent in the process of discovery raises the issue of race conditions. There exists the possibility of two new agents being started at the same time in the network which leads to a situation where the two new agents discover all the other agents in the network except each other. For example, let us assume that there exist three agents E1, E2, and E3 in the network and two new agents N1 and N2 are started at the same time with neighbours as E1 and E2 respectively.

- Both agents N1 and N2 contact their respective neigbouring agents E1 and E2 to request information about other agents in the network.

- Both agents N1 and N2 recieve the list containing information about E1, E2, and E3.

- The agent N1 does not recieve information about agent N2 from agent E1 since agent E1 is not aware of agent N2 yet. Similarly agent N2 is unaware of agent N1.

- when agents N1 and N2 recieve the information about other agents in the network they connect to the agents and exchange information.

At the end of the discovery process it can be seen that while the existing agents (E1, E2, and E3) become aware of the two new agents (N1 and N2) in the network, the new agents N1 and N2 do not have information about each other and will remain unaware of the other agents existance. This race condition can be avoided by configuring the agents to go through a two-stage discovery process in which the new agents connect to their *neighbour* agent, after the initial discovery is over, and request an agent table update. When the new agent N1 connects to its neighbour after the initial discovery it will receive an update with information about the other agent N2, thus avoiding the effects of the race condition.

3.2.6 Communication Between Agents

An ARROS agent can communicate with other agents in the network to share information about their network, information about any changes in the network, and request the other agent to respond to an intrusion. The different types of communications possible between agents are,

- 1 to 1
- 1 to many
- Broadcast

An agent can use one of the above modes of communication depending upon the type of action required. For example,

- agent A can request (1 to 1) agent B to block a certain connection
- agent A can request (1 to many) the agents that are on the border of the organizational network to block all connections to respond to a DDOS attack
- agent A can request (broadcast) all agents to block all traffic from/to a particular port to stop the spread of a worm or virus through the network.

3.2.7 Intelligent Distributed Response

The following scenario illustrates the need for responding to intrusions at a different node in the network:

When the intrusion is of the type that is utilizing much of the bandwidth, it is best to block the connection, by dropping all packets, as close to the source as possible. If the response is not close to the source, at A, then the connection continues to utilize the network resources of the host network until it reaches the victim machine, at J, where the packets are eventually dropped. This means that the network resources are being spent on routing packets through the network that are eventually going to be dropped. The agents decide on the point of response by combining the information from the IP traceback module and the information it has about other agents in the network.

Let us consider the example network, depicted in figure 3.14, which is six levels deep, with fourteen subnets A through N. Connection to the outside world is through subnet A. The attacker is in the outside world and the victim machine is on subnet J.

The subnets that have firewalls that are controlled by ARROS agents in the network are marked with an asterisk (*), which are subnets A, B, C, F, H, J, L, M, and N. The path of the intrusion from the attacker to the victim machine takes the path through subnets A-B-D-F-I-H-J. If the flow of traffic that occupies the bandwidth is from the victim to the attacker then the decision is to implement the rule locally. But if the flow of the bandwidth hogging traffic is from the attacker to the victim then the best place to block the intrusion and start dropping packets is at control-point A. The agent arrives at this decision by analyzing the intrusion information, the type of response required, the effect of the intrusions and the response on the network and the response capability of itself and the other agents in the network. Once the agent receives a request for response from the Decision Module, it analyzes the request and tries to choose the best control-point. It requests the path of the intrusion from the

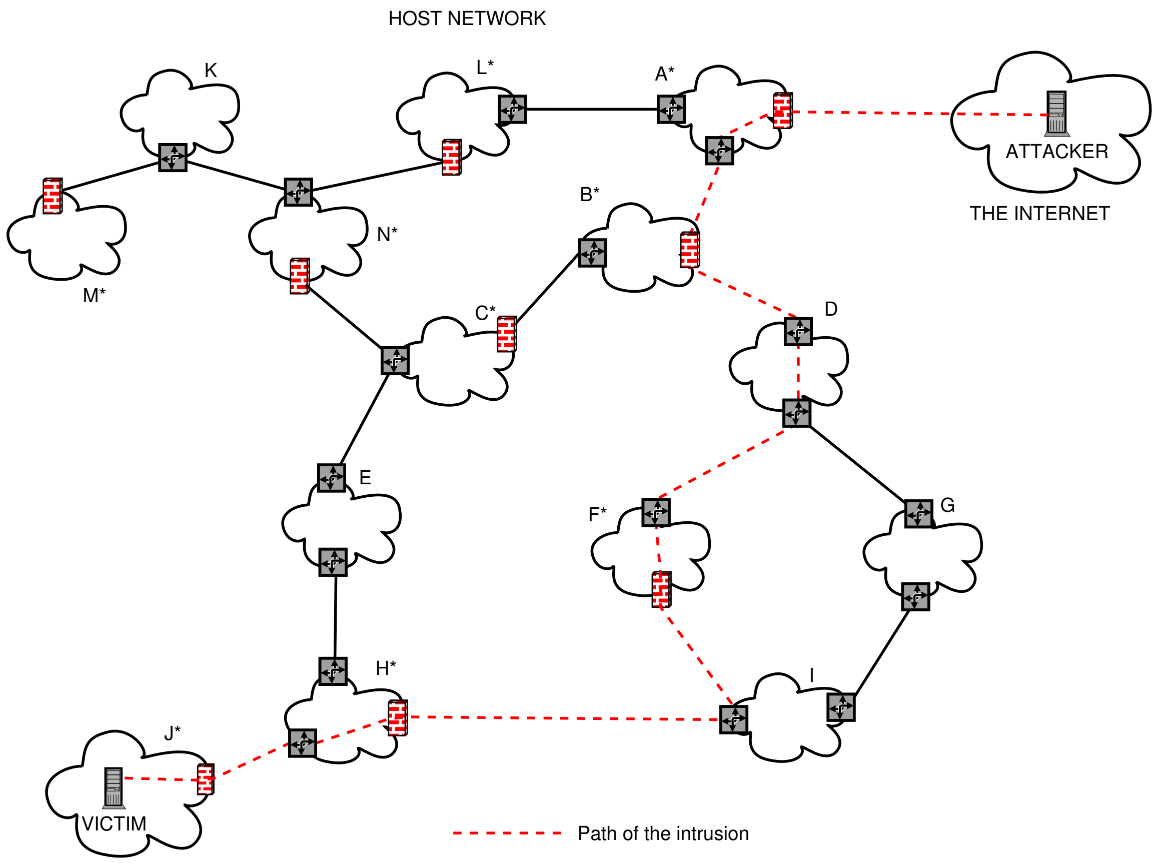


Figure 3.14. Tracking the Path of an Intrusion through the Network

connection tracking module and correlates that information with its peer agents table to get control-points along the path of the intrusion where response is possible. This procedure is shown diagrammatically in figure 3.15.

Now that the agent has all the possible control points , it can choose the best control point, or control points, based on the type of response, and the capabilities of the control point firewall. In this particular example it decides to implement the rule at A which is the closest point to the attacker where the agent has response capabilities. The agent can also:

- Request all the agents in the path to respond

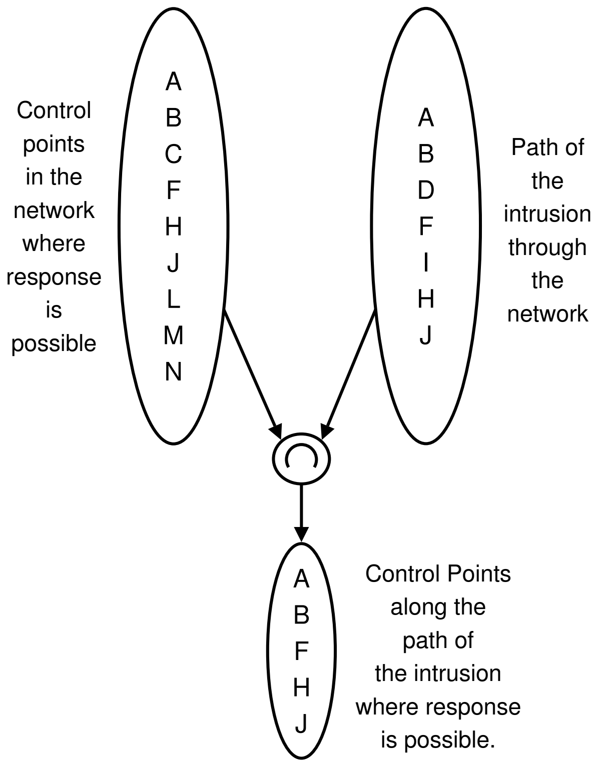


Figure 3.15. Selecting the Response Point

- Start working its way through the agent list in the specified order until it finds an agent that successfully blocks the connection.
- If it fails to find an appropriate response the agent implements the rule locally.

Chapter 4. Experimental Results

As described in Chapter 3, the ARROS agents provide automated IR capabilities to INBOUNDS. The agents network with each other during the discovery phase and share information about the network with each other. Agents also share information about intrusions and co-ordinate responses to the intrusions to provide distributed and intelligent response. The goals of the experiments were to test the ability of the ARROS agent's to successfully discover and network with other agents in the network, respond to intrusions locally, respond to intrusions in more than one location, and the ability to the system to perform these functions under stress. A test network was setup to run multiple ARROS agents to study the discovery times and response times of the agents under different network conditions. In this chapter, we present the results of our experiments with the prototype ARROS system.

4.1 Experiment Testbed

A test network consisting of eight computers was setup in a simple bus topology, as shown in Figure 4.1, using private IP addresses. Normal network usage was simulated by injecting packets into the test network using a packet generating tool. Traffic generation tool MGEN [6] was used to inject packets into the test network to simulate normal network utilization. Multi-Generator (MGEN)is an open-source software developed by the Naval Research Laboratory (NRL). MGEN was setup to inject packets into the network at a set rate to load the network with the required amount of background traffic.

The testbed consists of eight machines running Fedora Core 3, Linux version 2.6.9, on seven subnets. The hardware configuration of the computers used in the testbed is listed in Table 4.1. The six computers, labeled B through G, function as gateway

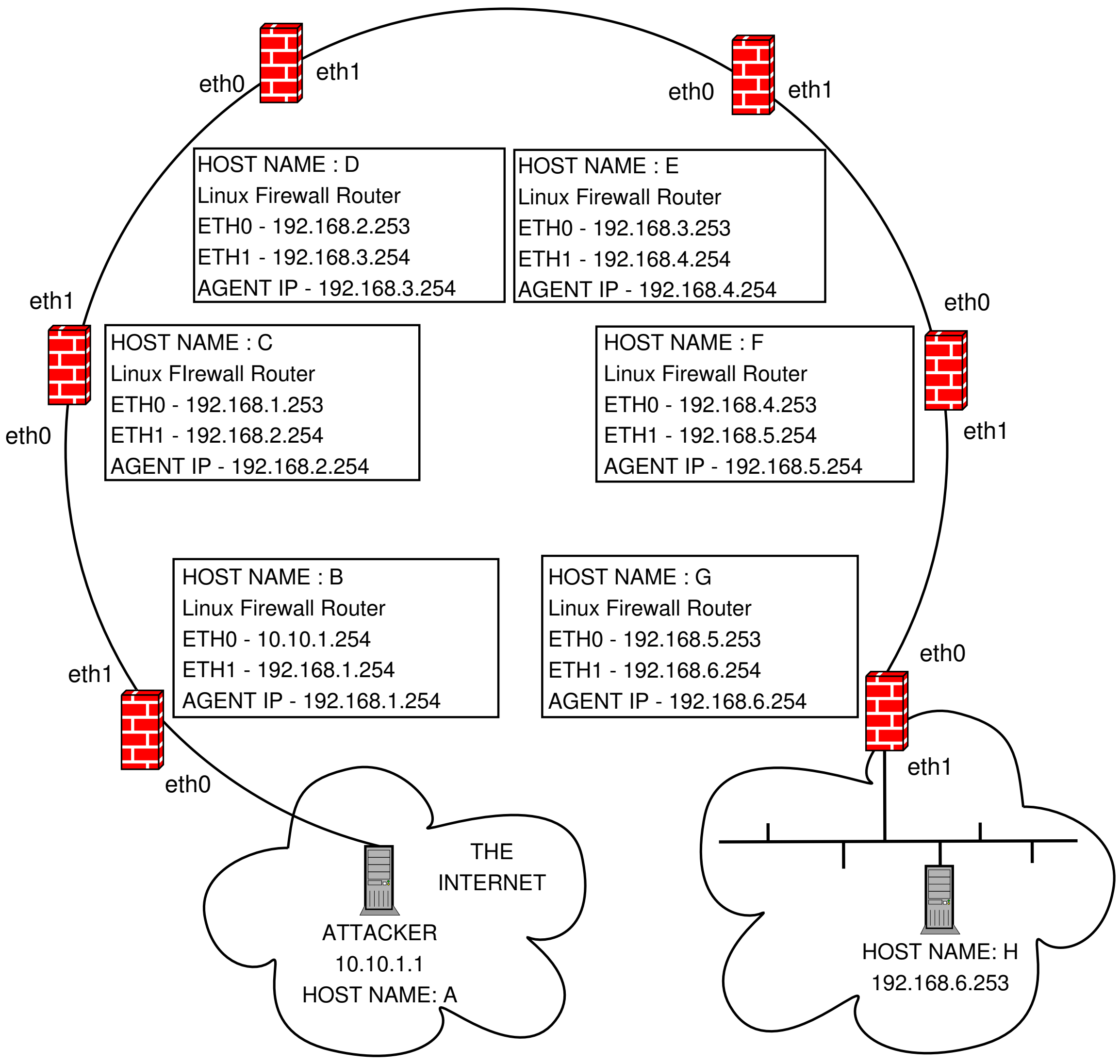


Figure 4.1. Test Network

routers for their respective subnets, and computer A functions as the attacker and computer H as the victim.

4.2 Experimental Results

The experiments were conducted to test the performance of the new INBOUNDS architecture and the ARROS agents and their ability to respond to intrusions in real-time, cope with failures in the network, cope with traffic congestions, speedy discovery of other agents, and provide distributed response.

The experiments are grouped into two categories, Discovery Time Tests and Response Time Tests. In the following sections we describe each type of experiment in detail, present the results of the tests, and follow them up with our analysis of the outcome.

4.2.1 Discovery Time Tests

The experiment is designed to measure the time taken by the agents in the network to learn about the existence of a new agent added to the network. When a new agent is started in a network where there already exist other ARROS agents, the new agent needs to gather information about the existing agents such as IP address, subnet information, the control points, and their capabilities. Similarly the agents in the network have to receive the same information about the new agent in the network so that the new ARROS agent becomes a part of the ARROS Agent network for sharing intrusion information and sharing response.

When a new agent is started in the network and there exist other agents in the network, then the new agent is given the IP address of one of the existing agents in the network. Without this one IP address the new agent is unable to discover the existing agents in the network and vice versa. When the new agent is started, and provided with an IP address of one other existing agent (neighbour) in the network, it immediately connects to the neighbour and requests the neighbour's copy of the agent table. Once the new agent has the neighbours agent table, it adds the information to its agent table and sends its own information to each of the agents in the table.

Table 4.1
Testbed Configuration Table

Name	CPU	Memory	Kernel Version	Interfaces	IP
A	Pentium Celeron 566Mhz	128M	2.6.9	eth0	10.10.1.1
B	Pentium Celeron 566MHz	128M	2.6.9	eth0 eth1	10.10.1.254 192.168.1.254
C	Pentium Celeron 566MHz	128M	2.6.9	eth0 eth1	192.168.1.253 192.168.2.254
D	Pentium Celeron 566MHz	128M	2.6.9	eth0 eth1	192.168.2.253 192.168.3.254
E	Pentium Celeron 566MHz	128M	2.6.9	eth0 eth1	192.168.3.253 192.168.4.254
F	Pentium Celeron 566MHz	128M	2.6.9	eth0 eth1	192.168.4.253 192.168.5.254
G	Pentium Celeron 566MHz	128M	2.6.9	eth0 eth1	192.168.5.253 192.168.6.254
H	Pentium Celeron 566MHz	128M	2.6.9	eth1	192.168.6.253

The tests were conducted with varying loads on the network, ranging from 0 percent network usage to 40, 70, and 95 percent network load.

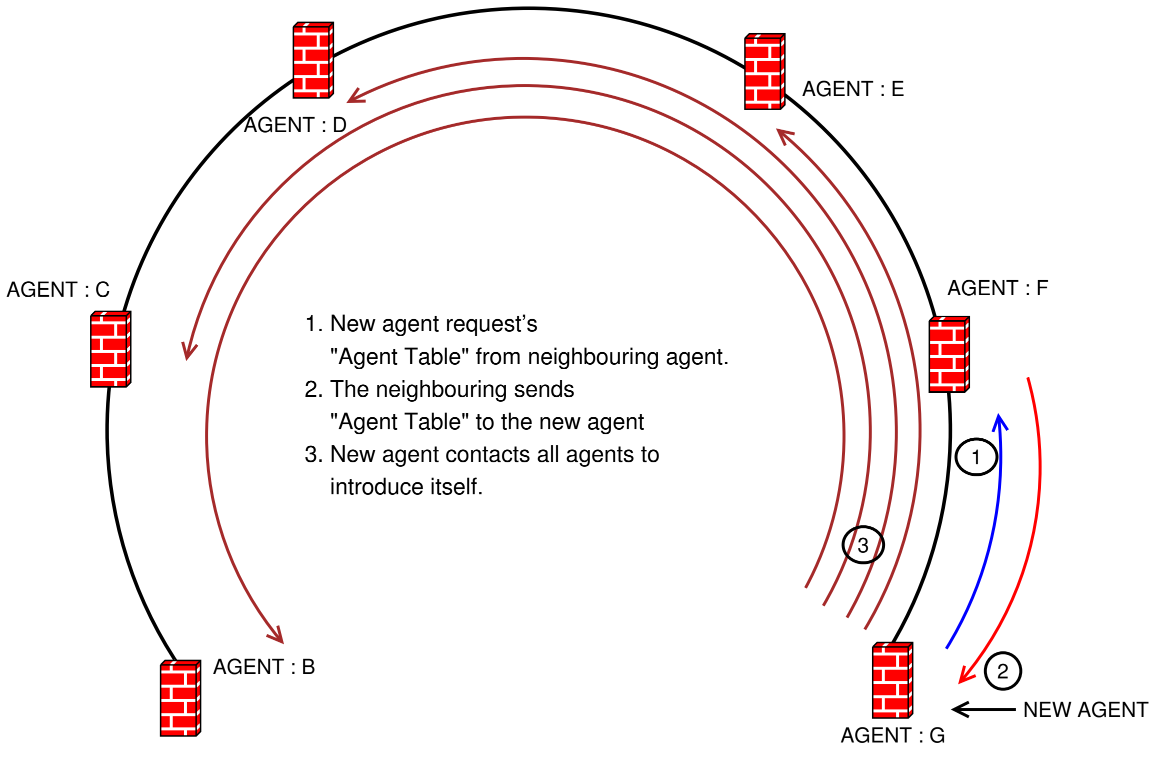

AGENT : D
AGENT : E
AGENT : C
AGENT : F
1. New agent request's "Agent Table" from neighbouring agent.
2. The neighbouring sends "Agent Table" to the new agent
3. New agent contacts all agents to introduce itself.
1
3
2
AGENT : B
AGENT : G
NEW AGENT

Figure 4.2. ARROS Agent Discovery

The following steps are performed under different network loads, as shown in Figure 4.2:

- MGEN is started on machine A, and a stream of UDP packets are sent to machine H to load the network to the required level.
- The ARROS agents are started on all the routers B, C, D, E, F, and G, in that order.

The log messages are analysed and the times taken by the ARROS agents to discover the new agent in the network are listed in Table 4.2. The time taken for

Table 4.2

Agent Discovery Time

Network Load	Average Time	MIN	MAX	Standard Deviation
0%	0.366 s	0.317 s	0.419 s	0.028 s
40%	0.528 s	0.476 s	0.565 s	0.022 s
70%	0.816 s	0.717 s	0.857 s	0.036 s
95%	1.144 s	0.978 s	1.236 s	0.061 s

discovery is calculated as the average of the discovery time for each agent where the discovery time is defined as the time difference between the time a new agent is started and the time an existing agent in the network receives information about the new agent. In each experiment, new agents are started, one after another, and the time delay between the starting of the new agent and the time each existing agent in the network becomes aware of the new agent is recorded and the average of all the discovery times is calculated.

It can been seen from Table 4.2 that the Discovery Time for the agents is less then 1 second, even when the network is loaded. The times listed in the table are inclusive of the time taken by the log messages to be sent from each ARROS agent to the Log Server that was setup up record the log messages, and hence are larger than the actual time it took for the agents to complete discovery.

4.2.2 Response Time Tests

The Response Time of the ARROS agents is an important factor that influences how successful the response is. The ability of the automated response to be implemented quickly is important in preventing the spread of fast attacks like the Sapphire worm [16]. The Response Time Test is designed to test the speed of response, the time from when the intrusion in detected by the IDS to the time when the response is implemented, and the ability of the ARROS system to provide distributed response. The tests are repeated under various network loads ranging from the total absence of network traffic to 40 percent, 70 percent, and 95 percent network utilization. During the course of each experiment the ARROS agent were tested for speed of response and tolerance to network failures. At certain points in time the agents in the network were disabled from responding to attacks to simulate the possibility of breakdown in the network due to failures that are inherent in networks, and the performance of the agent under these conditions and the ability to cope with the failures and the time to implement alternative responses was observed.

Four types of tests were conducted in this category:

- Single Point Local Response
- Single Point Remote Response
- Multi-point Response
- Broadcast Response

In the following sections we describe each type of experiment, the setup, and the analysis of the test results.

4.2.2.1 Single Point Local Response

When the intrusion is a single connection or multiple connections that requires the ARROS agent to respond by simply killing the connection and block it, the agent responds by simply modifying the firewall to kill the connection and block it. This kind of a simple response is used by the ARROS agents as a failsafe when distributed and remote responses fail.

In this response scenario, the response is implemented at the point of detection, locally, and only at that point. This type of a response is the best response in cases that involve a single intrusion connection. The response can be as simple as logging the connection or just killing the connection and preventing similar connection for the specified time period. Since the connection is either logged or terminated there is no need for distributed response or remote response since the connection is either logged or terminated.

The traffic generator, MGEN, is started on machine A and the traffic sink on machine H. ARROS agents are started on all the subnet gateway routers B, C, D, E, F, and G. After the agents discover themselves and the background traffic in the network was at the desired level, a simulated attack connection is made from machine A to machine H. Once the simulated attack connection is established an IDS alert is sent to the ARROS agent on the local gateway machine G requesting response. The time duration between the alert and the implementation of the response is observed and recorded.

In the second part of the test, once the simulated intrusion connection is established the response capability of the agent on G is disabled to simulate a breakdown in the system. The time taken by the ARROS agent to analyse, formulate and implement the response at an alternate location is observed and recorded. In this case the ARROS agent contacted each agent along the part of the intrusion, one by one, starting from the one closest to the victim and proceeding towards the attack source, until it was able to connect to an agent, agent F, that was able to respond to the intrusion.

Figure 4.3 shows the testbed setup, the location of the attacker, victim, the path of the intrusion through the network, the response point, and the process of intrusion response. The traffic generator was modified to generate background traffic up to 40, 70 and 95 percent of the network bandwidth, and in each case the test is carried out and the results recorded. The response times from the tests are tabulated in Table 4.3 and the throughput graph of the intrusion connection is shown in Figure 4.4. Figure 4.4 shows the effect of the response on the intrusion connection. The agent responds to the intrusion by inserting a rule into the firewall to block the connection for 30 seconds. After the response expires the agent rolls back the response to the

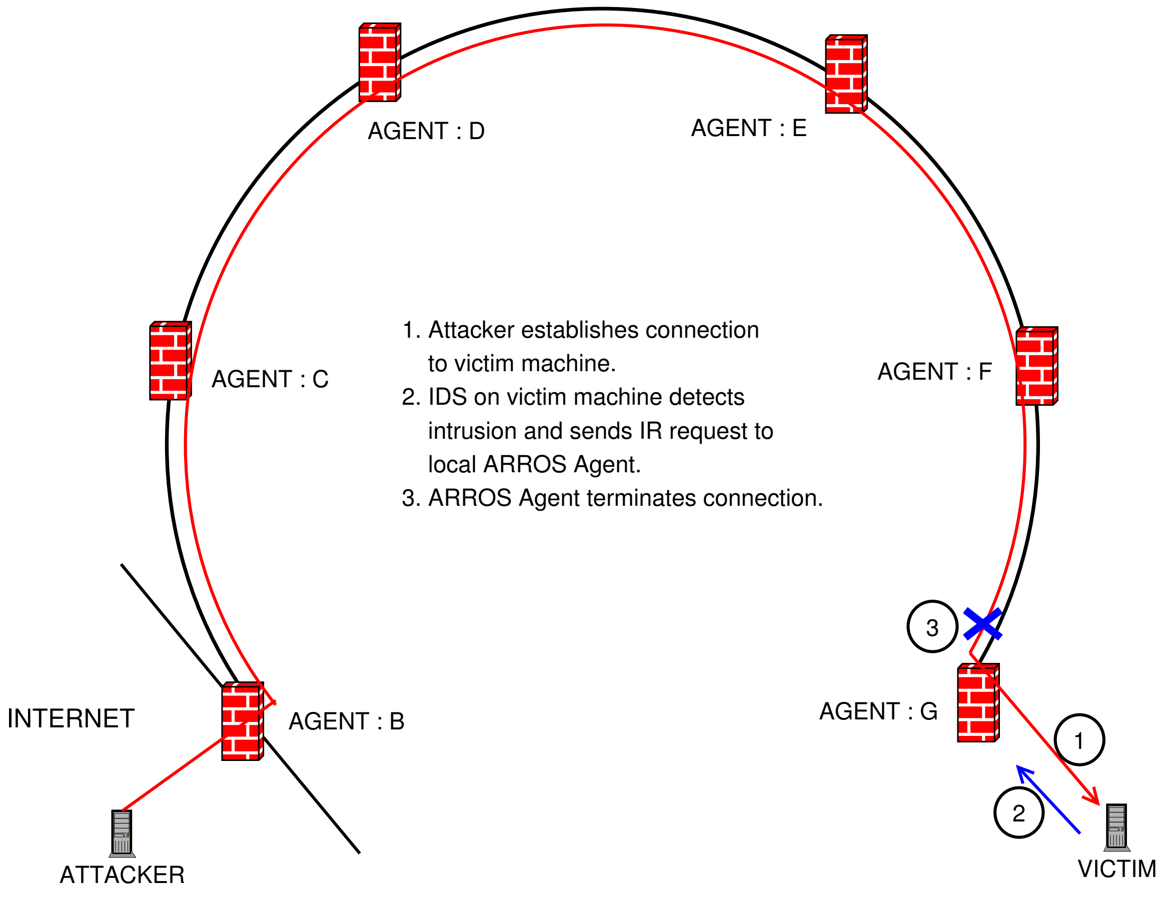

AGENT : D
AGENT : E
AGENT : C
AGENT : F
1. Attacker establishes connection to victim machine.
2. IDS on victim machine detects intrusion and sends IR request to local ARROS Agent.
3. ARROS Agent terminates connection.
3
INTERNET
AGENT : B
AGENT : G
1
2
ATTACKER
VICTIM

Figure 4.3. Single Point Local Response

intrusion connection which, at which point the connection resumes. The effectiveness of the ARROS IRS, in blocking the intrusion connection locally, is shown in Figure 4.4 which confirms that the intrusion connection is blocked from the time the intrusion response is requested and after the lifetime of the response, when the response is rolled back, the block is removed. It is clear from the figure that the agents are able to block connections locally for a given time period quickly, and without affecting other legitimate traffic.

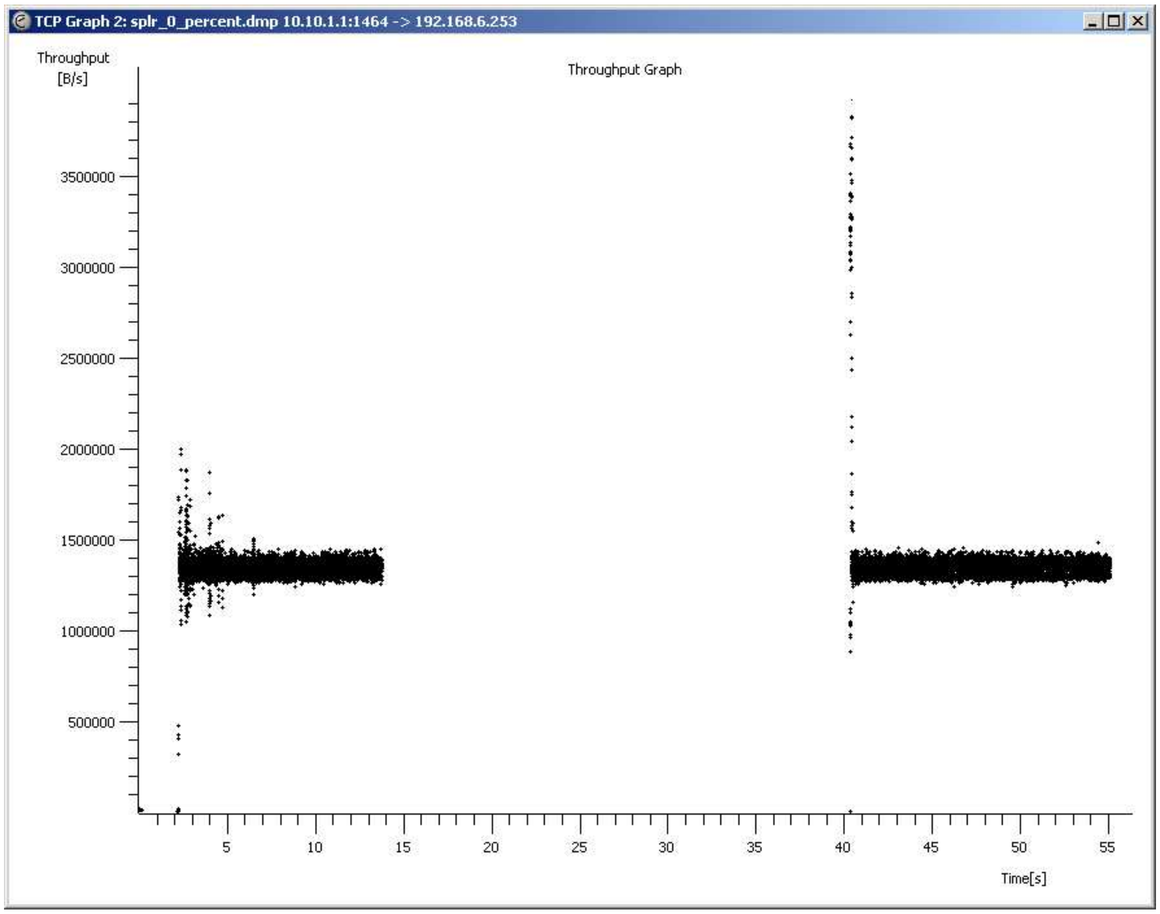


Figure 4.4. Single Point Local Response: 0% Background Traffic

Table 4.3
Single Point Local Response Times

Network Load (%)	Response Time (seconds)
0	0.006
40	0.009
70	0.012
95	0.016

4.2.2.2 Single Point Remote Response

Single point remote response is the type of response that is implemented at a single point, but is at a location that is not on the same subnet the victim is at or the subnet where the intrusion is detected. This type of response is typical for intrusion that consume the bandwidth, like file sharing. Also in some cases it is not possible to terminate certain types of traffic since it includes legitimate traffic. But a failure to respond leads to network congestion and failure. In such cases the response is to shape the traffic, limit its bandwidth, so that the network is no longer overloaded. A good example of this would be a situation where there is an email virus that is spreading, blocking all email traffic would affect legitimate email traffic while not responding to the threat might overload the network and disrupt other traffic. So in this case the most appropriate response would be to limit the bandwidth for email traffic.

Since the intrusion involves the use of network resources, the response to it is to limit the bandwidth, by dropping packets at set a rate. If the point of response is local, close to the victim, the packets would get dropped at the victim's subnet, after they have passed through the host network and used up its resources till that point. In this case the right point of response would be to start dropping the packets, limit its bandwidth, as soon as they come within the host network. So the point of response should be as close to the attack source as possible.

In this experiment the agents were started on all the subnets, and background traffic is generated to simulate required network utilization.

Once the agents are started and had discovered each other, a simulated bandwidth attack connection is made from machine A to machine H. Once the attack traffic starts to hog the bandwidth, an alert is sent to the agent on machine G, as shown in Figure 4.5. The time, including any network congestion delay, it took the agent on machine G to contact the agent closest to the source of the attack, machine B, to respond to the attack by limiting the bandwidth of the attack connection, is observed and

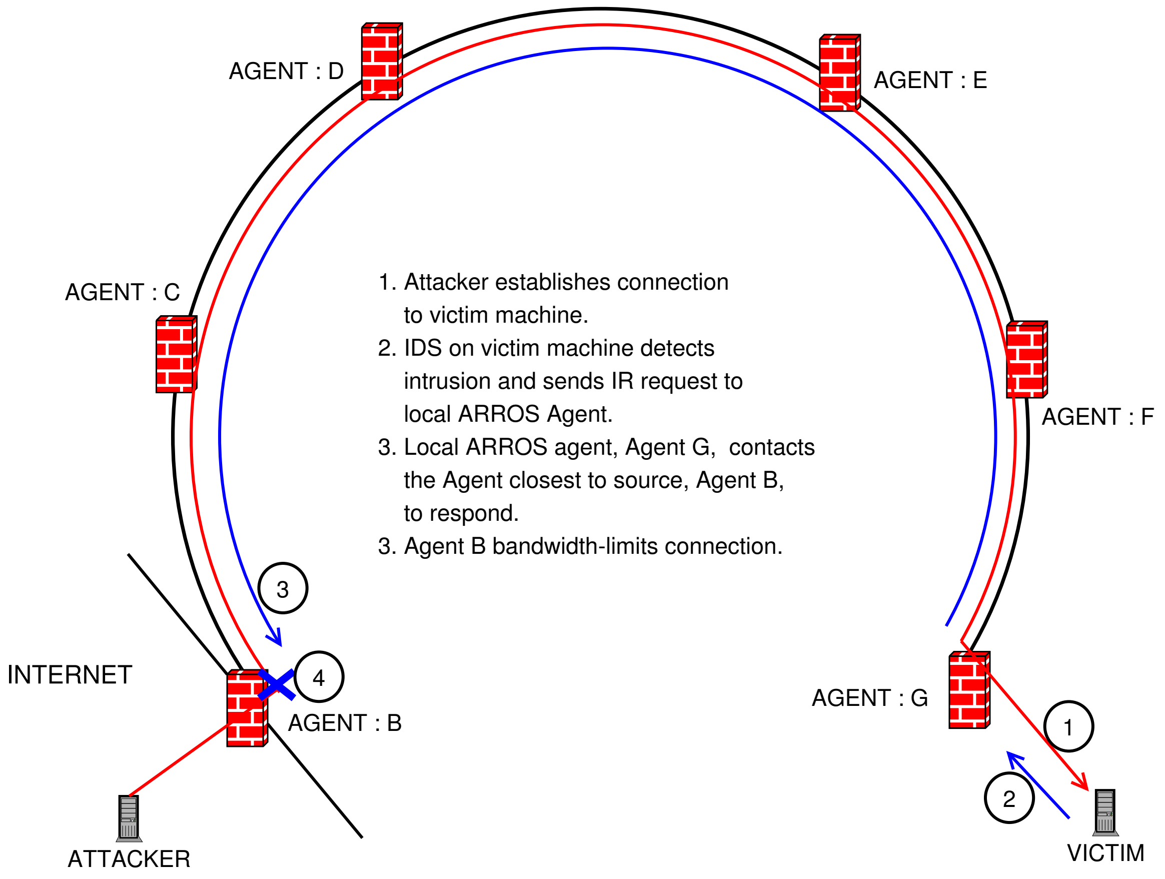

AGENT : D
AGENT : E
AGENT : C
AGENT : F
1. Attacker establishes connection to victim machine.
2. IDS on victim machine detects intrusion and sends IR request to local ARROS Agent.
3. Local ARROS agent, Agent G, contacts the Agent closest to source, Agent B, to respond.
3. Agent B bandwidth-limits connection.
3
INTERNET
4
AGENT : B
AGENT : G
1
2
ATTACKER
VICTIM

Figure 4.5. Single Point Remote Response

recorded. The changes in the network bandwidth usage, by the attack connection, is logged downstream on machine H. The Figure 4.6 shows a intrusion connection that is consuming the bandwidth in the network at which point it becomes an undesirable connection, a possible attack and an alert is sent to the agent to respond to the attack by shaping its traffic. The rule is inserted into the firewall to throttle the bandwidth of the connection, and the attack connections bandwidth is successfully throttled, shown by the drop in throughput of the intrusion connection in Figure 4.6.

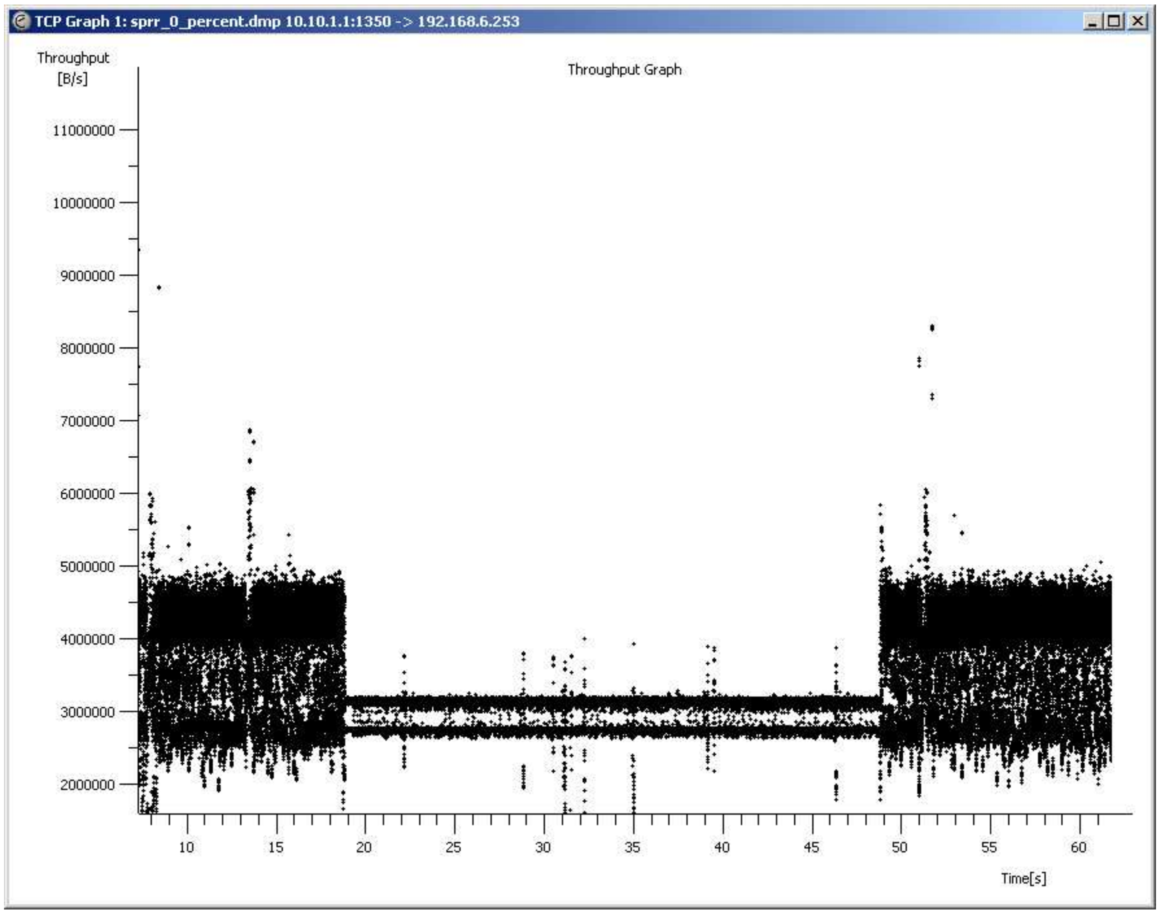


Figure 4.6. Single Point Remote Response: 0% Background Traffic

4.2.2.3 Multi-point Response

A multi-point response is required when only a few select agents are required to respond to the intrusion. An example of this type of response would be an response by:

- Agents along the path of the intrusion.
- All border router agents, in case of a DDOS.
- Only agent capable of a particular type of response, like traffic shaping.

Table 4.4
Single Point Remote Response Times: The graphs shows the successful blocking of the intrusion connection for the set time period, and the successful rollback of the response after its expiration

Network Load (%)	Response Time (seconds)
0	0.023
40	0.037
70	0.046
95	0.130

When the agent decides to implement a multi-point response, it refers to its agent cache and selects agents that meet the response criteria and contacts the agents to request them to respond to the intrusion. This is different from Broadcast Response, discussed in Section 4.2.2.4, where the ARROS agent contacts all agents it is aware of.

In this test the traffic generator, MGEN, is started on machine A and is run until the background traffic was at the required level. Simulated intrusion connections were made from machine A and B to the victim machine H. Similar connections were also made from machines C, D, E, and F to machine H to simulate legitimate connections that are similar to the intrusion connection. The purpose of the test was to show that the agents are able to provide multi-point response, based on the given criteria, so that they are able to respond to the intrusion at specific points in the network to block intrusions without disrupting similar looking legitimate connections.

In this experiment the traffic generator, MGEN, is started on machine A and the traffic sink is started on machine H. All the agents are started and a simulated attack connection is made from machine A to machine H. An alert is sent to the agent on machine G requesting it to implement a multi-point response using agents B and C.

Table 4.5
Multi-Point Response Times

Network Load	Average Time	MIN	MAX	Standard Deviation
0%	0.083 s	0.037 s	0.128 s	0.064 s
40%	0.085 s	0.051 s	0.120 s	0.049 s
70%	0.133 s	0.081 s	0.185 s	0.074 s
95%	0.295 s	0.198 s	0.392 s	0.137 s

The time taken by the select agents to implement response is observed and recorded. The throughput graphs of all the connections is shown in Figures 4.7, 4.8, 4.9, and 4.10. It can be seen from the figures that the connections from machines A and B are blocked by the ARROS agents and the rest of the connections are unaffected.

4.2.2.4 Broadcast Response

When an intrusion is deemed to be highly destructive, the response needs to be fast, widespread and restrictive to stop it from further spreading. A good example of one such incident would be the SQL Slammer worm that spread itself from one machine to another and infected all available machines in the world, in a matter of minutes. If such a worm were detected on the host network, the agent initiating the response would start a broadcast response request to all the agents in the network to block the worm, thereby containing the damage to the already infected subnets and stopping its spread to other subnets and the internet, and preventing anymore worm traffic from entering the network.

A broadcast response, as the name implies, involves the initiating ARROS agent sending requests for response to all the agents in the host network. Though the response does not involve any intelligent by the agent, in terms of exploring other

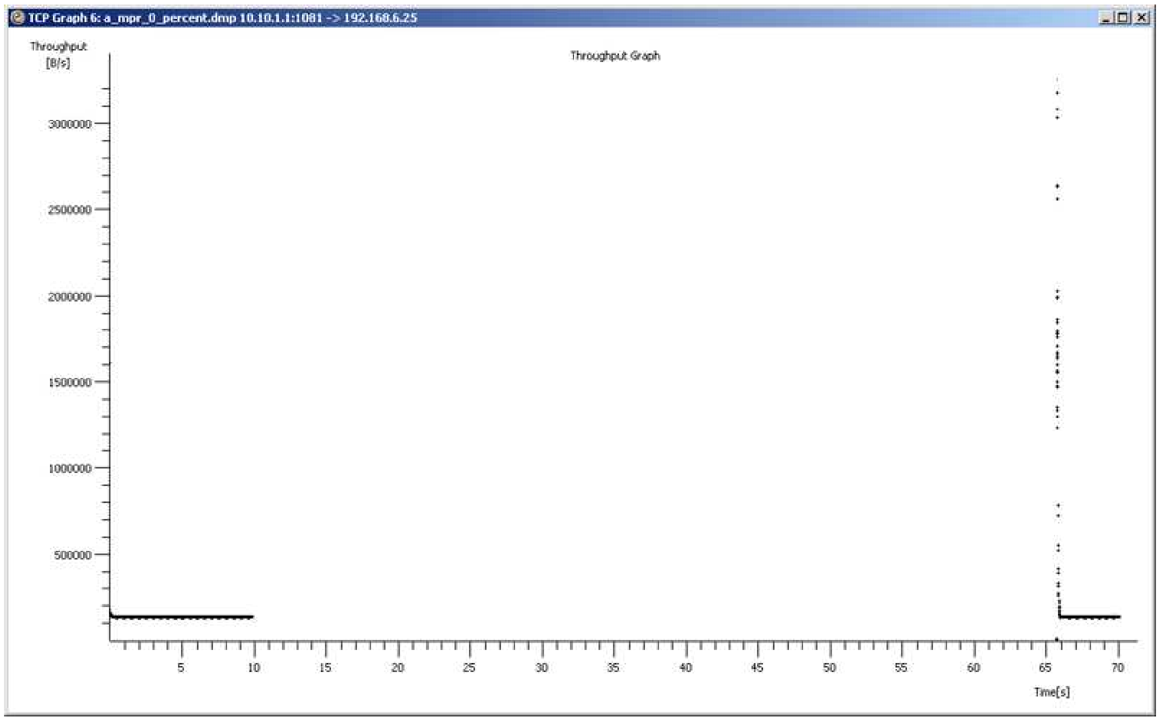


Figure 4.7. Multi-Point Response: 0% Background Traffic - Agent B: The connection from machine A, flagged as an intrusion, is successfully blocked by ARROS agent B

available response options, it still is the most effective and restrictive response to certain kinds of intrusions that are fast and highly destructive.

In this experiment all the ARROS agents are started, and the traffic generator, MGEN, is started on machine A and is set to generate varying amounts of traffic ranging from no traffic to 40, 70, and 95 percent of the total network bandwidth. Once the agents discovered themselves, an alert is sent to the agent on machine G to implement a Broadcast response, as shown in Figure 4.11. The time from the instant the agent received the alert, to the instant when all the agents on the network had successfully responded to the attack is observed and recorded. The test was carried out under varying network loads to observe the effect, if any, of network congestion on the response system. The response times are listed in Table 4.6. The throughput

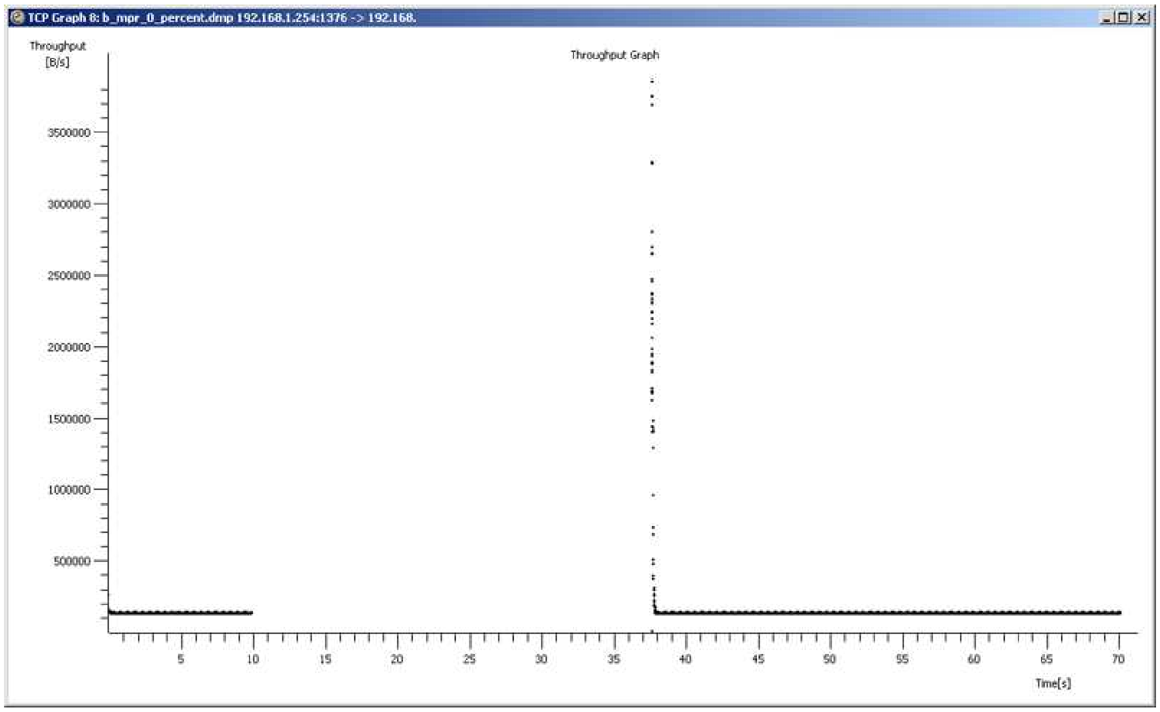


Figure 4.8. Multi-Point Response: 0% Background Traffic - Agent C: The connection from machine B that is flagged as an intrusion is successfully blocked by ARROS agent C

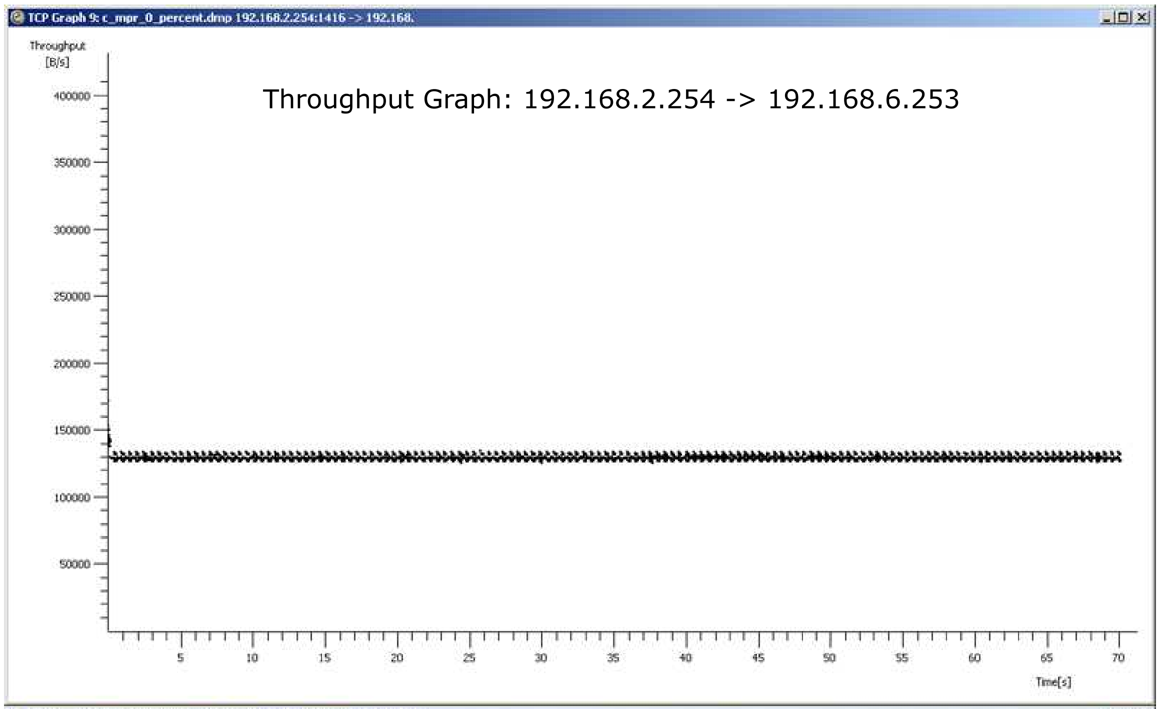
graph of all the intrusion connections is shown in Figures 4.12, 4.13, and 4.14, which shows the successful blocking of the intrusion connections by all the agents.

4.3 Analysis of Results

As described in Section 4.2 the new INBOUNDS architecture and the ARROS IRS prototype were run on a test-bed as a proof of concept and the time taken by the system to successfully respond to intrusions under varying network loads recorded.

The tests with the prototype ARROS system and the new architecture show that the new architecture and the ARROS system is able to:

- Respond quickly.
- Respond in multiple locations, distributed response.



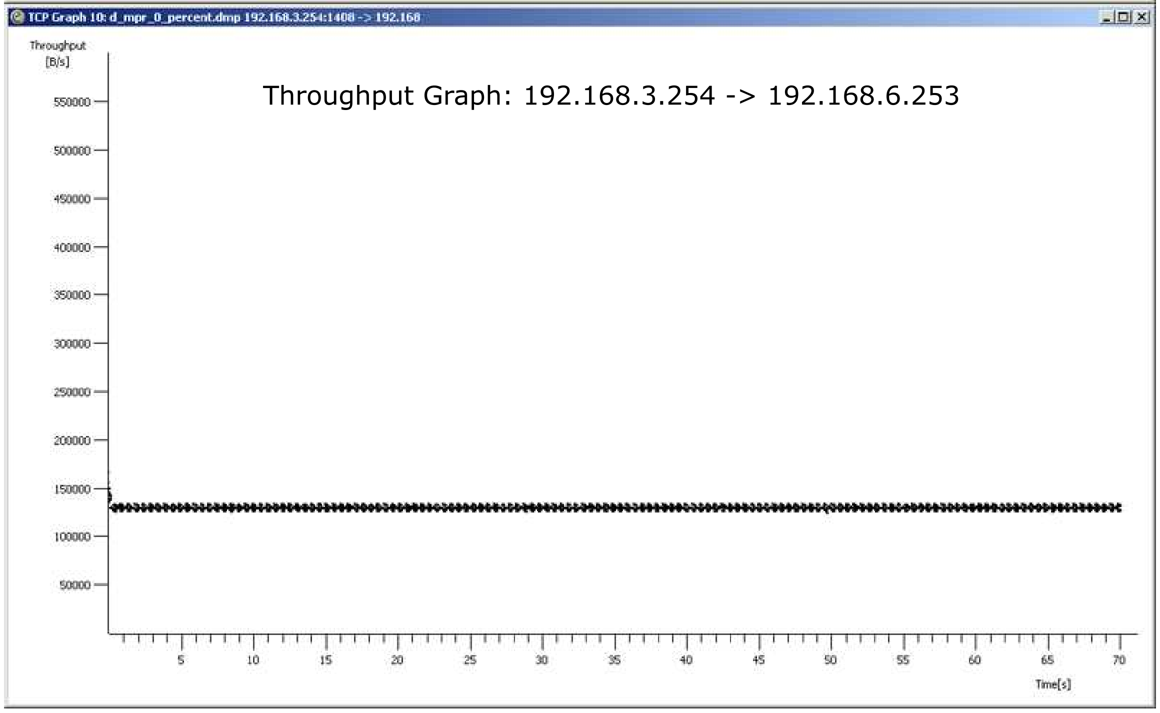
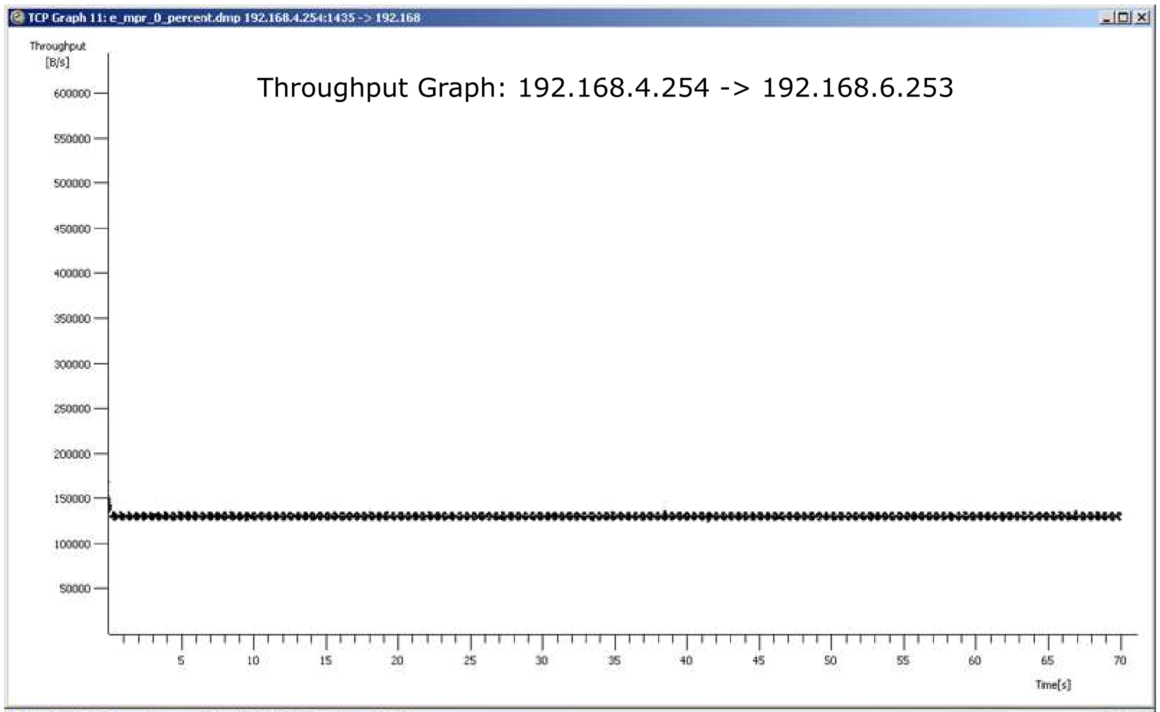


Figure 4.9. Multi-Point Response: 0% Background Traffic - Agent D and E: The connection from machines C and D that are legitimate connections, remain unaffected by the Multi-Point response



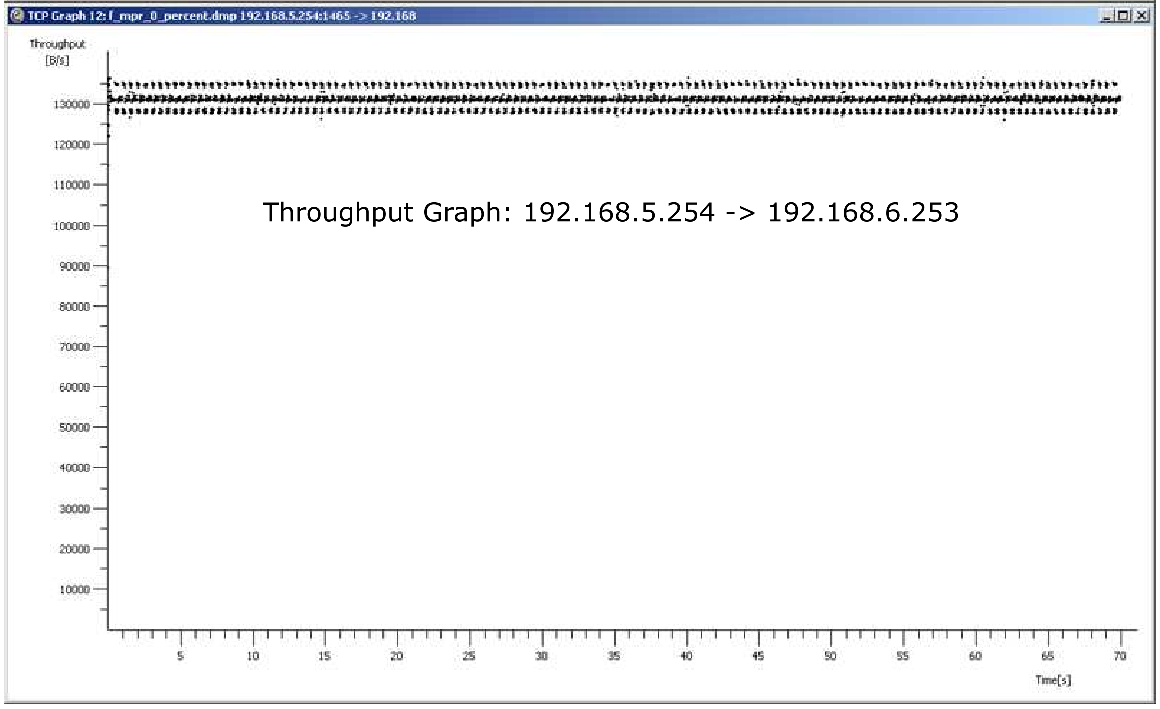


Figure 4.10. Multi-Point Response: 0% Background Traffic - Agent F and G: The connection from machines E and F that are legitimate connections, remain unaffected by the Multi-Point response

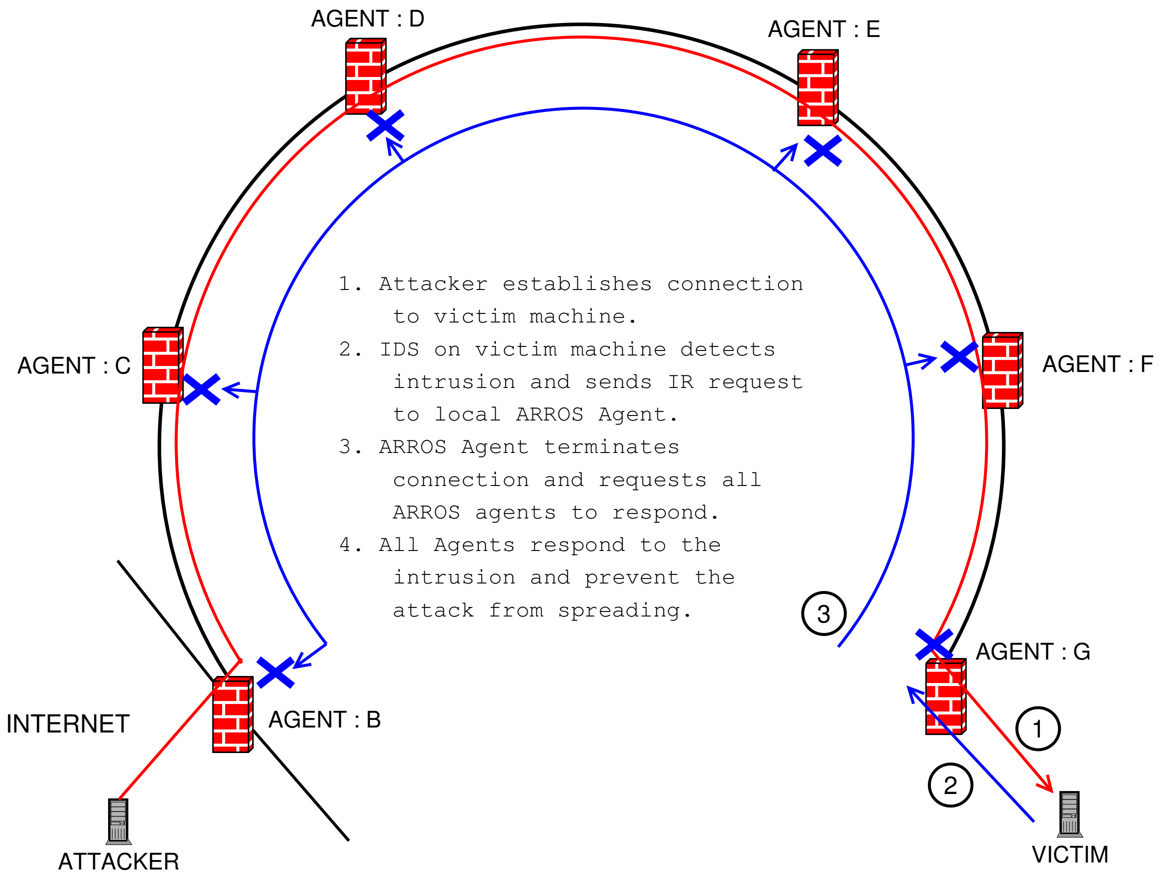

AGENT : D
AGENT : E
AGENT : C
AGENT : F
AGENT : G
AGENT : B
INTERNET
ATTACKER
VICTIM
1
2
3
1. Attacker establishes connection to victim machine.
2. IDS on victim machine detects intrusion and sends IR request to local ARROS Agent.
3. ARROS Agent terminates connection and requests all ARROS agents to respond.
4. All Agents respond to the intrusion and prevent the attack from spreading.

Figure 4.11. Broadcast Response

Table 4.6

Broadcast Response Time

Network Load	Average Time	MIN	MAX	Standard Deviation
0%	0.092 s	0.005 s	0.182 s	0.066 s
40%	0.156 s	0.014 s	0.305 s	0.108 s
70%	0.259 s	0.020 s	0.489 s	0.176 s
95%	0.327 s	0.022 s	0.733 s	0.265 s

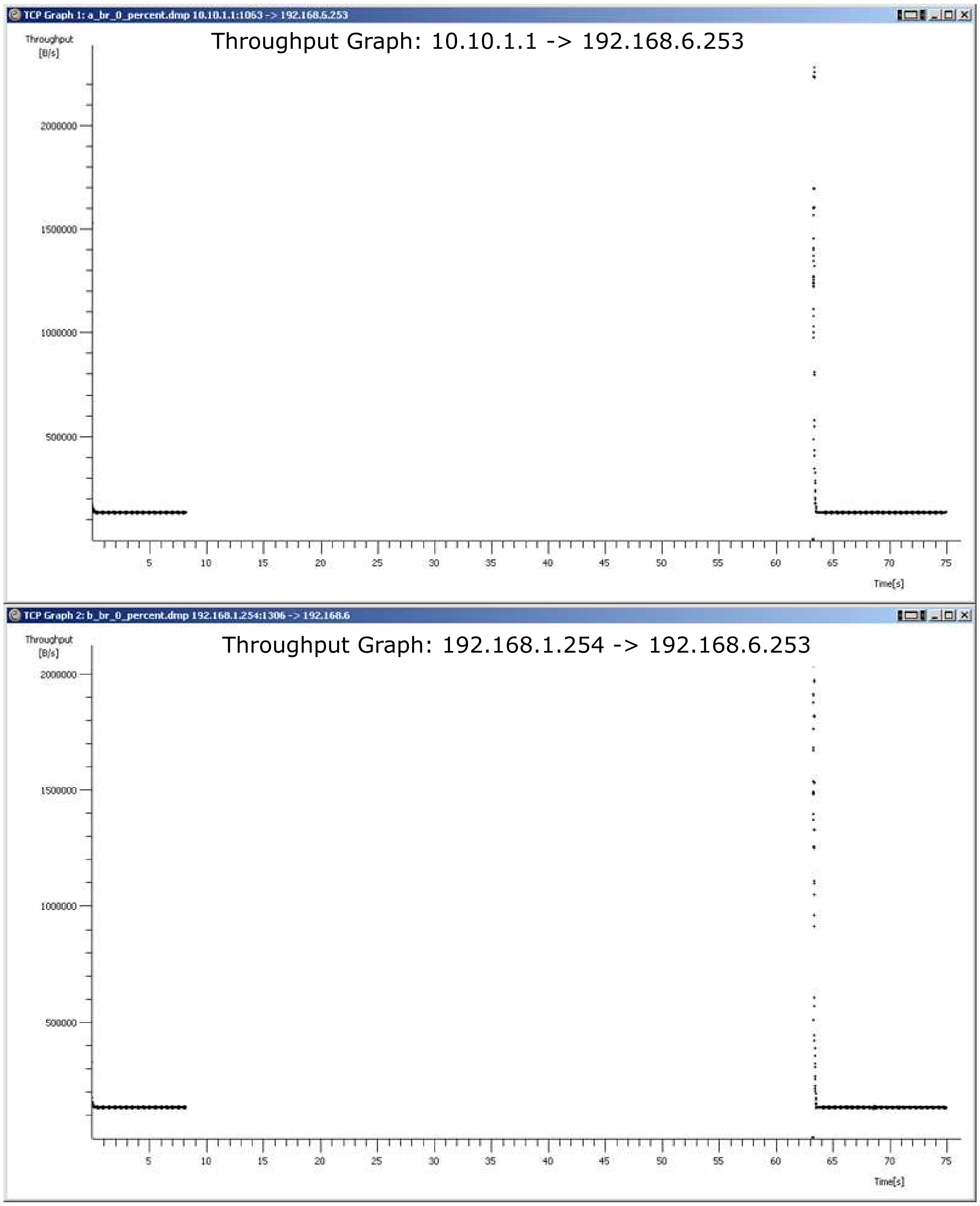
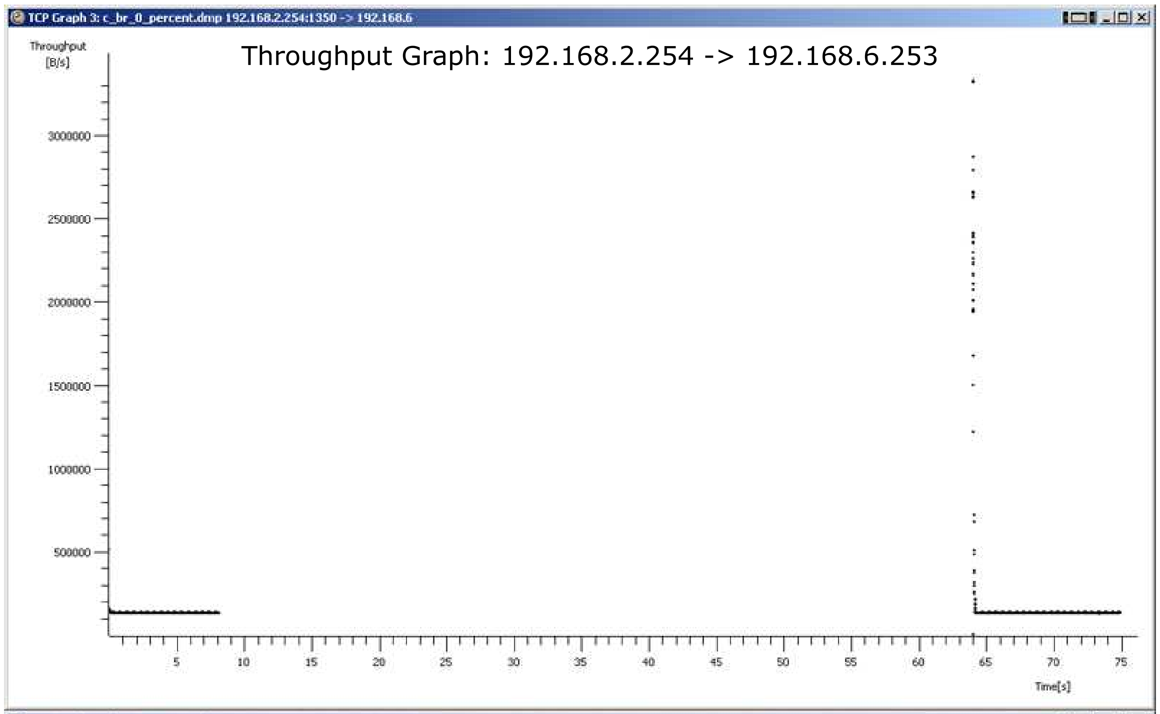


Figure 4.12. Broadcast Response: 0% Background Traffic - Agent B and C: The intrusion connections from machine A and B are successfully blocked by agents B and C



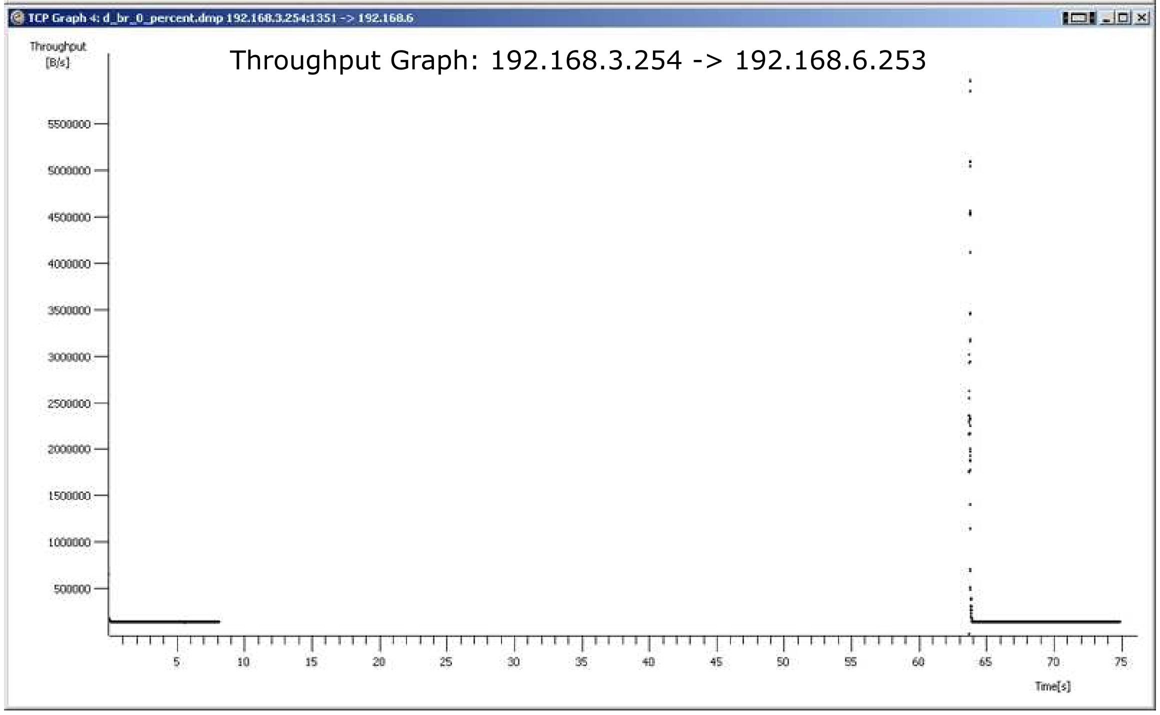


Figure 4.13. Broadcast Response: 0% Background Traffic - Agent D and E: The intrusion connections from machine C and D are successfully blocked by agents D and E

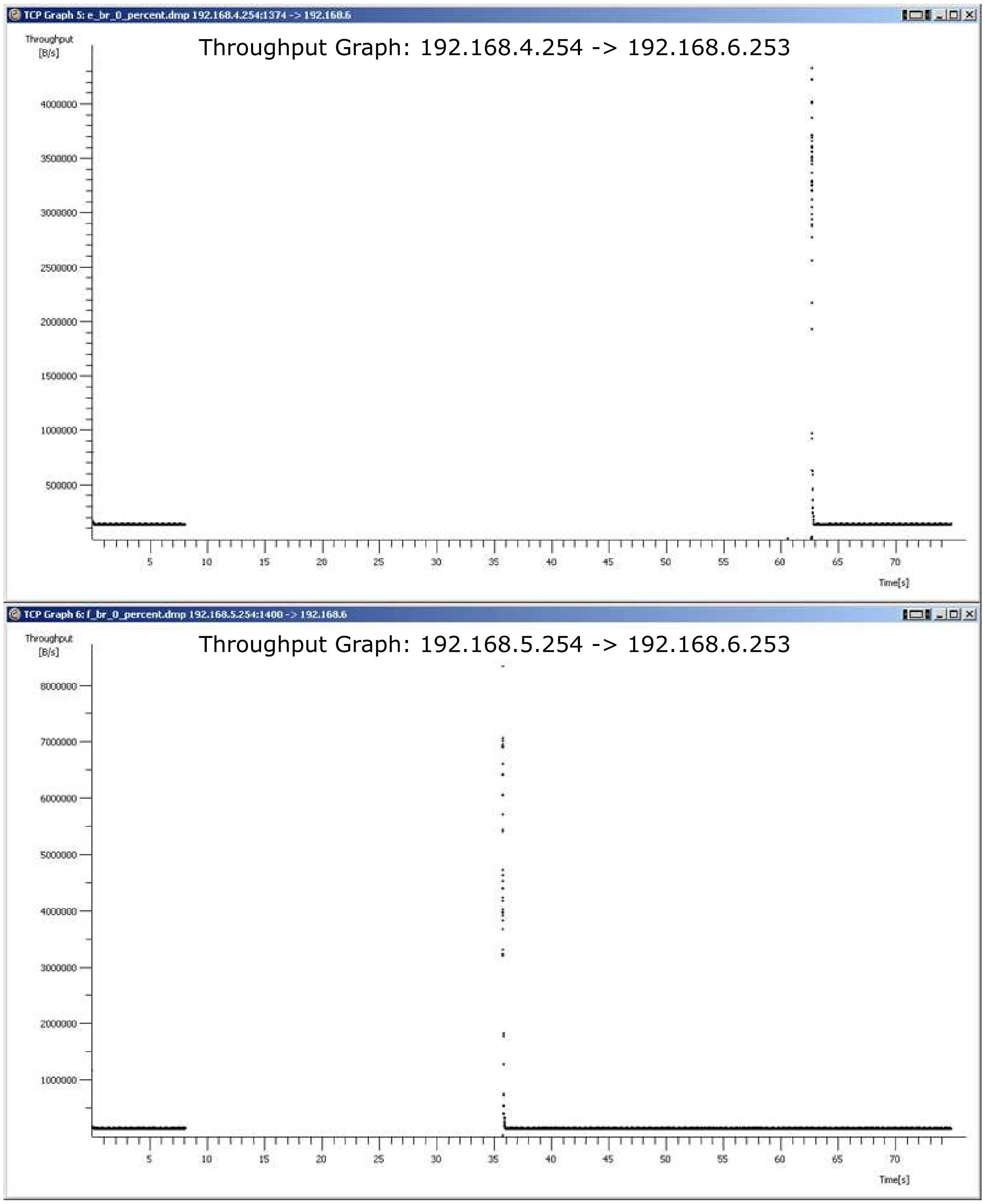


Figure 4.14. Broadcast Response: 0% Background Traffic - Agent F and G: The intrusion connections from machine E and F are successfully blocked by agents F and G

- Rollback responses automatically.
- Recover from and adapt to failures in response.

From Table 4.2 it is evident that the discovery time required by ARROS agents is, on average, less than 1 second even under overloaded network conditions. This is a important factor in deciding how effective a response is in stopping the intrusion. The time taken by agents to communicate with each other is reduced by using priority queueing, which ensures that the agents are able to network with new agents and provide fast response.

The response times of the agents listed in Table 4.3, Table 4.4, Table 4.5, and Table 4.6 for each type of test conducted. The ability of the IRS to respond quickly to intrusions is important as it reduces the time gap available to the intruder to carry out the attack. It is evident from the tests that the ARROS agents are able to provide response times in the order of microseconds even when the network is under load. In each of the throughput graphs it is seen that the agents are successful in blocking the connection and shaping it bandwidth. The agents are also successful is rolling back the response after the stipulated period of time.

The agents respond to intrusions in a time sensitive manner to alleviate any adverse effects to the host network due to false positives. One of the biggest drawbacks of automated intrusion response is the ability of the intruder to use the IRS to turn off legitimate traffic , DOS attack, thereby using the IRS itself to mount attacks. But the default behavior of the ARROS system is to give the intrusion responses a definite lifetime, once the lifetime of a particular intrusion response is over the response in rolled back. This behavior of the ARROS agent ensures that false positives do not affect the host network over long periods of time and keeps the firewall ruleset from bloating while discouraging script-kiddies and casual hackers who loses interest after a few attempts to hack the network.

In each of the experiments, when a failure was introduced in the response, the agents were able to recover from and adapt to the failure in response and were able to implement the response at an alternate location.

Chapter 5. Conclusion

In this chapter, we present a summary of the design and operation of the new INBOUNDS architecture and the ARROS system, its strengths and weaknesses, and follow it up with recommendations for future work.

The ARROS IRS and the proposed architecture provide distributed intrusion detection and response by using multiple intrusion response agents at various different point in the host network. The the IDS, IRS, and the Intrusion Tracker work together to detect the intrusion, track its path through the host network and respond to the intrusion by terminating the connection or by limiting its bandwidth. Experiments with the new INBOUNDS architecture and the ARROS system has shown that the agents are able to:

- Respond quickly.
- Respond in multiple locations, distributed response.
- Automatic Rollback ability.
- Recover from and adapt to failures in response.

5.1 Advantages and Disadvantages

The ARROS IRS for INBOUNDS is able to respond successfully to intrusion in less than 1 second from the time the intrusion is detected. The agents also rollback the firewall changes once the response time expires, thereby keeping the firewall from bloating and reducing network traffic disruption times due to false positives. The ARROS agents also communicate with each other and the Intelligence Module to exchange information on success/failure of responses which provides feedback to the

Intelligence Module to improve future responses. The “Intelligence Module” still depends on the administrator to report false positives. The agents require initial setup to be able to discover existing agents in the host network, and if the agent is unable to communicate with its specified neighbour-agent during the start-up phase then it is cut-off from the rest of the ARROS agents.

5.2 Future Work

The current implementation of the ARROS IRS is a proof of concept and does not include a robust firewalling mechanism. Also the system currently lacks a working “Intelligence Module”. It will be interesting to test the system with an “Intelligence Module” in place that will be able to correlate IDS real-time alerts from multiple IDSs, past IDS alerts, and IDS confidence metrics to generate a high-level intrusion response scenario that can be implemented by the ARROS IRS.

Bibliography

[1] Ethereal. `http://www.ethereal.com`.

[2] Access Control Lists: Overview and Guidelines.
Cisco Access Control List Documentaion.

[3] Internet Domain Survey 2003, Internet Software Consortium.
`http://www.isc.org/ds`.

[4] IPFWADM Project. `ftp://ftp.xos.nl/pub/linux/ipfwadm/`.

[5] Linux IP Firewalling Chains. `http://www.netfilter.org/ipchains`.

[6] MGEN: The Multi-Generator Toolset. `http://mgen.pf.itd.nrl.navy.mil/`.

[7] AAIRS: An Intrusion Response Taxonomy and its Role in Automatic Intrusion Response. In *Proceedings of the 2000 IEEE Workshop on Information Assurance and Security* (United States Military Academy, West Point, NY, 2000).

[8] CERT/CC Statistics, CERT Coordination Center, 2003.
`http://www.cert.org/stats/`.

[9] Balupari, R. Real-Time Network-Based Anomaly Intrusion Detection. Master's thesis, Ohio University, Mar. 2002.
`http://portal.acm.org/citation.cfm?id=903866.903869`.

[10] Blanton, E. Tcpurify: TCP Packet Sniffer, Sept. 2002.
`http://irg.cs.ohiou.edu/~eblanton/tcpurify`.

[11] Bruggeman, C., Welch, L., Gillen, M., Ostermann, S., Yellapragada, R., and Karunanidhi, K. Secure-RM: Security as a QoS Constraint in Real-Time Middleware. In *The Proceedings of The 2003 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'03)* (2003).

[12] Bykova, M. INBOUNDS Tcptrace Module: Packet Analyzer, Sept. 2003.
`http://cidds.cs.ohiou.edu/~inbounds/downloads.shtml`.

[13] Cohen, F. Simulating Cyber Attacks, Defenses, and Consequences. `http://all.net/journal/ntb/simulate/simulate.html`.

[14] Janakiraman, R., Waldvogel, M., and Zhang, Q. Indra: A Peer-to-Peer Approach to Network Intrusion Detection and Prevention. In *WETICE '03: Proceedings of the Twelfth International Workshop on Enabling Technologies* (Washington, DC, USA, 2003), IEEE Computer Society, p. 226.

[15] Jr, C. A. C., Hill, J. M., and R, J. A Methodology for Using Intelligent Agents to Provide Automated Intrusion Response. In *IEEE Workshop on Information Assurance and Security* (jun 2000).

[16] Moore, D., Paxson, V., Savage, S., Shannon, C., Staniford, S., and Weaver, N. Sapphire/Slammer Worm, January 2003. `http://www.cs.berkeley.edu/~nweaver/sapphire/`.

[17] Netfilter. Netfilter/IPTABLES. `http://www.netfilter.org/`.

[18] Ostermann, S. Tcptrace: TCP Dump File Analysis Tool, Apr. 2003. `http://tcptrace.org`.

[19] Porras, P. A., and Neumann, P. G. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In *1997 National Information Systems Security Conference* (oct 1997). `http://www.sdl.sri.com/papers/emerald-niss97/`.

[20] Ramadas, M. Real-Time Network-Based Anomaly Intrusion Detection. Master's thesis, Ohio University, June 2002. `http://www.ohiolink.edu/etd/view.cgi?ohiou1049472005`.

[21] R.Heady, G.Luger, A.Maccabe, and M.Servilla. The Architecture of a Network Level Intrusion Detection System. Tech. rep., University of New Mexico, 1990.

[22] SANS. SANS Intrusion Detection FAQ: Network Intrusion and use of Automated Responses. `http://www.sans.org/resources/idfaq/auto_res.php`.

[23] Tcpdump. Tcpdump. `http://www.tcpdump.org`.

[24] Toth, T., and Kruegel, C. Evaluating the Impact of Automated Intrusion Response Mechanisms. In *18th Annual Computer Security Applications Conference* (Dec. 2002). `http://wwwcsif.cs.ucdavis.edu/~balepin/new_pubs/response_cost.pdf`.

[25] White, G., Fisch, E., and Pooch, U. Cooperating Security Managers: A Peer-based Intrusion Detection System. *IEEE Network 10*, 1 (Jan. 1996).

[26] YELLAPRAGADA, R. Probabilistic Model for Detecting Network Traffic Anomalies. Master's thesis, Ohio University, March 2004. http://www.ohiolink.edu/etd/view.cgi?acc_num=ohiou1088538020.

Appendix A. ARROS Communication Module Message Formats

The ARROS Communication Module handles allthe inter-agent and intra-agent communications. When the communication module connects to another entity, it sends a standard header first followed by the actual data that needs to be sent. Similarly when the communication module receives a connection it expects to receive the header followed by the actual data. In both cases the transmission of data is an optional, and the whole communication can just be the transmission of the header. In the following sections we describe in detail the message formats used by the Communication Module for data exchange.

A.1 Header

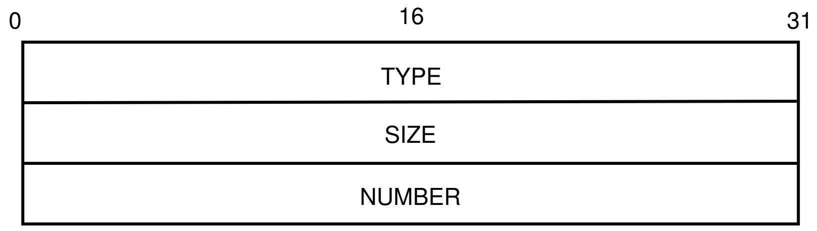


Figure A.1. Header Message Format

The ARROS Communication module sends and recieves the header in the beginning of each communication to let the reciever know the type and amount of data that is to follow. The *Type* field in the header indicates the type of the communication being initiated. The list of allowed values and their meanings are shown in table A.1.

Table A.1
Type field in Header Message Format

Type	Communication
0	Intrusion Alert.
1	Request for Tracking an intrusion.
2	Path of the intrusion through the host network.
3	Request for Intrusion Response.
4	Status of the Request for Intrusion Response received.
5	Request for Agent Table.
6	Agent Table entries.
7	Update for Agent Table.

The *size* and *number* fields in the header indicate the size of each unit of data that follows and the total number of units being sent, respectively.

A.2 Agent Table Request

When the ARROS Communication module sends a request to its neighbouring ARROS agent, to request a copy of the neighbours Agent Table, it sends a request with the header *type* field set to 5, and the *size* and *number* fields set to zero. The Agent Table request consists of just the header and is not followed by any data.

A.3 Agent Table Entries

Each Agent Table entry has information about an ARROS agent and information about the firewall devices that it controls. The format of the Agent Table entry and the Firewall Device information are shown in figures A.3 and A.2.

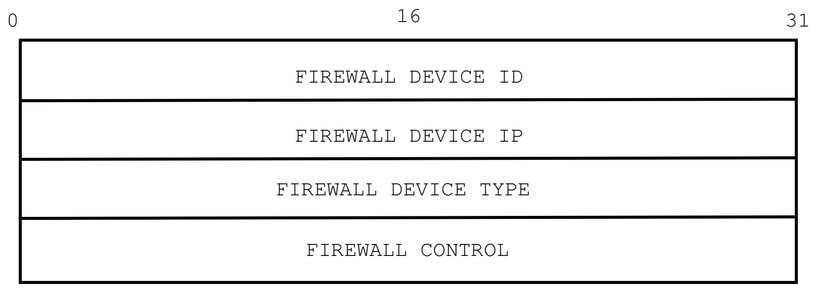


Figure A.2. Firewall Device Information Message Format

A.4 Remote Intrusion Response Request

The Remote Intrusion Response Request message consists of information about the intrusion and the response to be implemented. The *TTL* field indicates the lifetime of the response. The *Request Type Field* in the message indicates the type and scope of the response. It is a 32 bit field. The first 16 bits are unused. Bits 16 to 23 indicate the scope of the response and bits 24 to 31 indicate the type of response to be implemented. The format of the *Request Type Field* and the Remote Intrusion Response Request message are shown in figures A.5 and A.4.

A.5 Status of Remote Intrusion Response Request

When an agent recieves a request for intrusion response from another agent, it implements the response and returns a status message, to indicate the success/failure of the response, back to the requesting agent. The format of the status message is shown in figure A.6. The status is set to 0 for failure and 1 for success.

A.6 Request Intrusion Tracking

When an agent needs the path of an intrusion it sends the intrusion information to the tracker. The format of the message is shown in figure A.7.

A.7 Path of Intrusion Through the Host Network

When an agents requests the path of an intrusion, the tracker returns the path of the intrusion through the host network, if available. The format of the message is

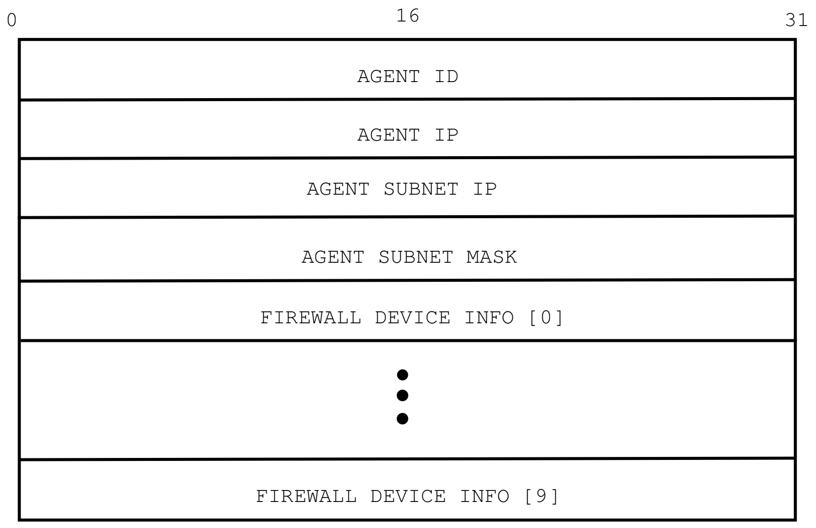


Figure A.3. Agent Table Entries Message Format

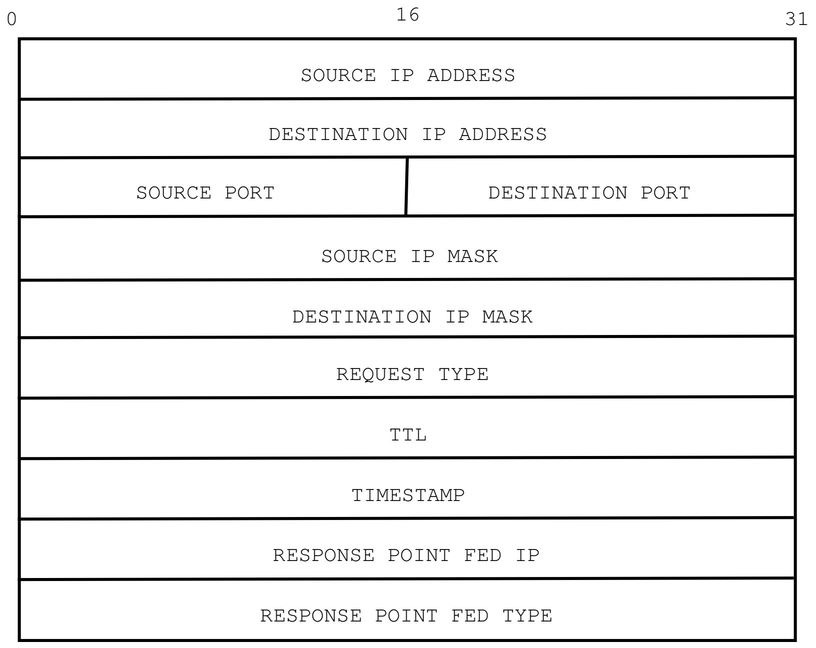


Figure A.4. Intrusion Response Request Message Format

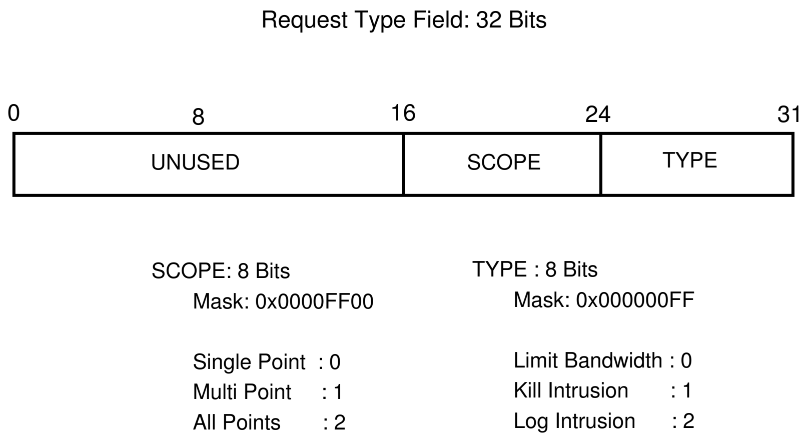


Figure A.5. Request Type Field Format

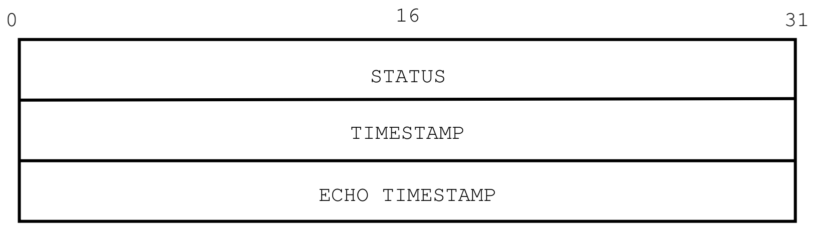


Figure A.6. Intrusion Response Request Status Message Format

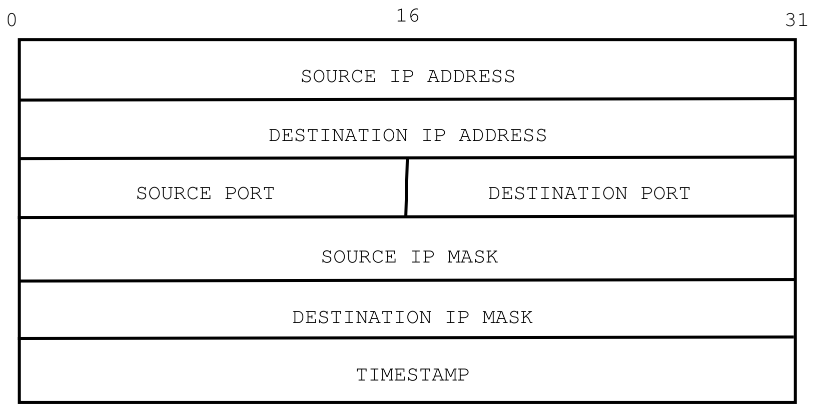


Figure A.7. Intrusion Track Request Message Format

shown in figure A.8. It is a list of 9 IP addresses, one for each HOP, starting with the border router.

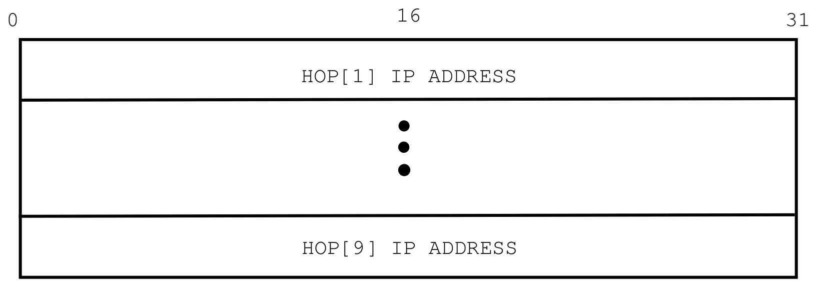


Figure A.8. Intrusion Track Message Format

Appendix B. Response Time of ARROS Agents

Table B.1 lists the individual response times for each of the agents in the Broadcast Response tests, under varying network loads.

Table B.2 lists the individual response times for each of the agents in the Multipoint Response tests, under varying network loads.

Table B.1
Individual Agent Response times during Broadcast Response Tests

Agent\Network Load	0%	40%	70%	95%
B	0.185	0.305	0.484	0.733
C	0.146	0.240	0.400	0.517
D	0.110	0.182	0.311	0.347
E	0.081	0.123	0.211	0.234
F	0.040	0.069	0.120	0.110
G	0.006	0.014	0.020	0.022

Table B.2
Individual Agent Response times during Multi-Point Response Tests

Agent\Network Load	0%	40%	70%	95%
B	0.037	0.051	0.081	0.198
C	0.128	0.120	0.185	0.392

Appendix C. Scripts used by ARROS to control the Firewall

C.1 IPTABLES preparation script.

```
#!/bin/bash

#reset everything in Iptables

iptables=/sbin/iptables

${iptables} -t filter -F

${iptables} -t mangle -F

${iptables} -X

${iptables} -Z


echo 1 > /proc/sys/net/ipv4/ip_forward


${iptables} -A OUTPUT -t mangle -p tcp --dport 9999 -j MARK \

--set-mark 1

${iptables} -A PREROUTING -t mangle -p tcp --dport 9999 -j MARK \

--set-mark 1

${iptables} -A OUTPUT -t mangle -p tcp --dport 9998 -j MARK \

--set-mark 1

${iptables} -A PREROUTING -t mangle -p tcp --dport 9998 -j MARK \

--set-mark 1

${iptables} -A OUTPUT -t mangle -p tcp --sport 9999 -j MARK \

--set-mark 1

${iptables} -A PREROUTING -t mangle -p tcp --sport 9999 -j MARK \

--set-mark 1
```

```
${iptables} -A OUTPUT -t mangle -p tcp --sport 9998 -j MARK \
--set-mark 1
${iptables} -A PREROUTING -t mangle -p tcp --sport 9998 -j MARK \
--set-mark 1

${iptables} -A OUTPUT -t mangle -p tcp --dport 9997 -j MARK \
--set-mark 1
${iptables} -A PREROUTING -t mangle -p tcp --dport 9997 -j MARK \
--set-mark 1

${iptables} -A OUTPUT -t mangle -p tcp --sport 9997 -j MARK \
--set-mark 1
${iptables} -A PREROUTING -t mangle -p tcp --sport 9997 -j MARK
--set-mark 1
```

C.2 HTB Traffic Shaping Setup script..

```
#!/bin/bash
tc=/sbin/tc
INTERFACEPREFIX='eth'
INTERFACENUMBER=$1
ONE=1
INTERFACESUFFIX=${INTERFACENUMBER}
INTERFACE=${INTERFACEPREFIX}${INTERFACESUFFIX}
INTERFACEBW=$2
BWBOTTLENECK=$3
UNIT='mbit'
MAX_VALUE=${INTERFACEBW}
MAXBOTTLENECK_VALUE=${BWBOTTLENECK}
THROTTLEBY=4
```

```
THROTTLE_VALUE=$((${MAXBOTTLENECK_VALUE} / ${THROTTLEBY}))
MAX=${MAX_VALUE}${UNIT}
MAXBOTTLENECK=${MAXBOTTLENECK_VALUE}${UNIT}
THROTTLE=${THROTTLE_VALUE}${UNIT}

#DELETE THE ROOT
${tc} qdisc del dev ${INTERFACE} root handle 1:

#ADD THE ROOT
${tc} qdisc add dev ${INTERFACE} root handle 1: htb default 12
${tc} class add dev ${INTERFACE} parent 1: classid 1:1 htb rate \
${MAX} ceil ${MAX}

#ARROS - low latency, high priority
${tc} class add dev ${INTERFACE} parent 1:1 classid 1:10 htb rate \
${MAX} ceil ${MAX} quantum 7000 prio 0

#THROTTLED TRAFFIC - low priority, 1/4 of the bottleneck rate \
${tc} class add dev ${INTERFACE} parent 1:1 classid 1:11 htb rate \
${THROTTLE} ceil ${THROTTLE} quantum 2000 prio 7

#ALL OTHER TRAFFIC THAT IS UNMARKED
${tc} class add dev ${INTERFACE} parent 1:1 classid 1:12 htb rate \
${MAX} ceil ${MAX} quantum 7000 prio 1

#WHICH PACKETS GO TO WHICH CLASS
${tc} filter add dev ${INTERFACE} parent 1: protocol ip prio 0 handle \
1 fw classid 1:10
```

```
${tc} filter add dev ${INTERFACE} parent 1: protocol ip prio 1 handle \
2 fw classid 1:11
```

C.3 IPTABLES Modifying Script.

```
#!/bin/bash
#VARIABLE DECLARATIONS
iptables=/sbin/iptables
action=$1
localip=$2
localmask=$3
if [ $4 -eq 0 ]
    then
        localport=" "
else
    localport="--dport $4"
fi
remoteip=$5
remotemask=$6
if [ $7 -eq 0 ]
    then
        remoteport=" "
else
    remoteport="--sport $7"
fi
requesttype=$8
protocol=$9

#SCRIPT EXECUTION STARTS HERE
if [ $8 -eq 1 ]
```

```
    then
        #LOG
        if [ "$1" = "INSERT" ]
    then
        #write the rule here
        echo "Inserted IPTables Logging rule"
else
            #write the rule here
    echo "Deleted IPTables Logging rule"
fi
exit 1
fi
if [ $8 -eq 2 ]
    then
        #TERMINATE
        if [ "$1" = "INSERT" ]
    then
    ${iptables} -A FORWARD -p ${protocol} -s ${remoteip}/${remotemask} \
    ${remoteport} -d ${localip}/${localmask} ${localport} -j DROP
    echo "Inserted IPTables terminating rule"
else
    ${iptables} -D FORWARD -p ${protocol} -s ${remoteip}/${remotemask} \
    ${remoteport} -d ${localip}/${localmask} ${localport} -j DROP
    echo "Deleted IPTables terminating rule"
fi
exit 1
fi
if [ $8 -eq 3 ]
```

```
    then
        #LIMIT
        if [ "$1" = "INSERT" ]
    then
        ${iptables} -A PREROUTING -t mangle -p ${protocol} -s \
${remoteip}/${remotemask}  ${remoteport} -d \
${localip}/${localmask}  ${localport} -j \
MARK --set-mark 2
echo "Inserted IPTables limiting rule"
else
    ${iptables} -D PREROUTING -t mangle -p ${protocol} -s \
    ${remoteip}/${remotemask}  ${remoteport} -d ${localip}/${localmask} \
    ${localport} -j MARK --set-mark 2
    echo "Deleted IPTables limiting rule"
fi
exit 1
fi

echo "|           ERROR: UNKNOWN RESPONSE TYPE            |"
exit 0
```