



REYKJAVÍK UNIVERSITY
HÁSKÓLINN Í REYKJAVÍK

Controlling the Effects of Anomalous ARP Behaviour on Ethernet Networks

Daði Ármannsson
Master of Science
January 2007

Supervisor:
Gísli Hjálmtýsson
Professor

Reykjavík University - Department of Computer Science

M.Sc. Thesis



CONTROLLING THE EFFECTS OF ANOMALOUS ARP BEHAVIOUR ON ETHERNET NETWORKS

by

Daði Ármannsson

Thesis submitted to the Department of Computer Science at Reykjavík
University in partial fulfillment of the requirements for the degree of
Master of Science

January 2007

Thesis Committee:

Dr. Gísli Hjálmtýsson, Supervisor
Professor, Reykjavik University, Iceland

Dr. Laurent Mathy
Assistant Professor, Lancaster University, UK

Björn Brynjúlfsson
System Architect, Cadec Global LLC, USA

Copyright
Daði Ármannsson
January 2007

Abstract

There is a growing interest in large-scale Ethernet-based local and metropolitan area networks. A significant reason for their proliferation is the relatively simple manner in which they can be configured and deployed. An elementary service on these networks is the Address Resolution Protocol (ARP). This protocol is used to determine the link-layer address of a host given its network-layer identifier, and uses the broadcast capability of Ethernet to determine these mappings.

In this thesis, we present a thorough analysis of three sizable Ethernet-based local area networks that are representative of current Ethernet usage. In particular, we investigated scalability issues associated with the use of broadcast based control channel protocols on these networks, e.g. ARP. We find that under normal circumstances, the use of these protocols does not impose scalability limitations on Ethernet networks, especially not as the available bandwidth on those networks increases. However, due to poorly configured or malicious software, ARP could cause performance issues to arise. We propose a scheme that limits the effects of these issues, further enabling the scalability of Ethernet-networks under these extreme circumstances. The scheme extends the functionality of link-layer switches, such that they probabilistically drop ARP requests from hosts that generate an unusually large amount of broadcast messages. We evaluate the scheme through the use of simulation and an actual implementation on the Intel IXP1200 network processor platform.

Útdráttur

Nýlega hefur áhugi á stórum staðarnetum byggðum á Ethernet-tækni aukist til muna. Ein megin ástæðan fyrir velgengni þeirra er það hversu auðvelt það er að byggja slík net. Ein af undirstöðuþjónustum þessara neta er ARP prótókollurinn, sem notaður er til að varpa IP-vistföngum yfir í vélbúnaðar vistföng. Notast ARP við útvarpsstuðning Etherneta til að nálgast þessar varpanir.

Í þessari rígerð fjöllum við um ýtarlega rannsókn á þremur stórum netum byggðum á Ethernet-tækni sem við teljum vera einkennandi fyrir þau Ethernet sem nú eru í notkun. Sérstaklega beinum við athygli okkar að áhyggjum af skalanleika Etherneta, tengdum prótókollum sem byggja á notkun Ethernet-útkasts (t.d. ARP). Niðurstöður okkar benda til þess að við eðlilegar kringumstæður sé ekki ástæða til að hafa áhyggjur af notkun útvarps á Ethernetum, sérstaklega í ljósi þess að framboð bandvíddar er stöðugt að aukast. Ákveðnar tegundir hugbúnaðar geta þó ollið minnkuðum afköstum Etherneta, t.d. netormar. Við kynnum aðferðir til að minnka áhrif þessa hugbúnaðar. Með notkun þessara aðferða aukum við skalanleika Etherneta enn frekar. Aðferðirnar útvíkka virkni hefðbundins Ethernet búnaðar, þannig að hann hendir ARP fyrirspurnum með ákveðnum líkum. Líkur þess að ARP fyrirspurn sé hent eru háðar því hversu ört sendandi fyrirspurnarinnar sendir ARP fyrirspurnir. Við höfum mælt aðferðirnar með notkun hermílikana, auk þess sem við höfum útfært þær og mælt á IXP1200 netörgjörvanum frá Intel.

To my parents, for without their encouragement and support, I would never have had the chance to write this thesis.

Acknowledgements

The work presented in this thesis was in part funded by E-NEXT European Network of Excellence and the Iceland's National Research Council. The work on the ARP management scheme was done in cooperation with Dr. Paul Smith and Dr. Laurent Mathy of Lancaster University's InfoLab21. I would like to thank Andy Myers of Carnegie Mellon University for his help and my friend and colleague, Gunnar Kristjánsson, for his valuable input and help with resolving technical issues. Finally, I would like to thank my supervisor for his guidance and the members of the thesis committee for their valuable comments.

Publications

Some of the material presented in this thesis has appeared in other publications. In *Controlling the Effects of Anomalous ARP Behaviour on Ethernet Networks* (Ármannsson, Smith, Hjalmtýsson, & Mathy, 2005), we present parts of the ARP behavior analysis, as well as an initial version of the ARP management scheme. A further paper focusing on the additional ARP management approaches presented in this thesis is in progress. The work presented in (Hjalmtýsson & Ármannsson, 2005) focuses on network analysis and is part of a broader research topic which this paper is a contribution to.

Contents

1	Introduction	1
2	Background	5
2.1	The Address Resolution Protocol	6
2.1.1	Operation of ARP	6
2.1.2	ARP Table	8
2.1.3	Other ARP Uses	8
2.1.4	Malicious Uses of ARP	9
2.2	Internet Worms	10
2.2.1	History	10
2.2.2	Purposes and Effects	12
2.2.3	Mitigation	12
3	Network Analysis	15
3.1	ARP Broadcast Distribution	16
3.2	ARP Traffic Patterns	19
3.3	The Effect of Malicious and Misconfigured Devices	20
4	Managing Anomalous Behavior	25
4.1	Probabilistic Dropping Scheme	26
4.2	Scopes	28
4.3	Stream Identifiers	29
4.4	Permutations	30
5	Evaluation	33
5.1	ARP Throttling	34
5.2	Rate of dropped requests	34
5.3	Normal vs. anomalous requests	39
5.4	Retransmissions	42
6	Implementation on the Intel IXP1200 Network Processor	47
6.1	Intel IXP1200	47
6.2	Evaluation	48
7	Conclusions	51
	Bibliography	53

A Design

55

List of Figures

2.1	Address resolution performed by host s for the protocol address of host t . Furthermore, host u receives the request and updates its address mapping for the protocol address of host s	7
2.2	A) A host performs address resolution by broadcasting an ARP request. The switch in the center of the topology copies the frame to three ports and three end-systems process the request. B) The target replies using unicast. The switch forwards the reply frame on a single port and only the originator of the process processes the reply.	7
3.1	The number of devices broadcasting ARP requests at a given second over one day, taken from $N3$	16
3.2	The number of ARP request broadcast at a given second over one day, taken from $N3$	17
3.3	Burstiness characteristics of ARP traffic	18
3.4	Traffic patterns showing the relationship between ARP sources and targets. This graph plots data from network $N1$ with the axes representing device identifiers. .	20
3.5	The increase in request rate for hosts infected with a worm scanning the local network for vulnerable machines.	21
3.6	The infection dissemination and ARP volume caused by infected devices for $0 \leq i \leq 40$	22
4.1	Different scheme scopes. The number of end-systems that affect each other is dependent on the size of the scope.	26
5.1	Number of received requests and forwarded requests per second on network $N2$ using approach SP	35
5.2	Number of received requests and forwarded requests per second on network $N2$ using approach SH	35
5.3	Number of received requests and forwarded requests per second on network $N2$ using approach PH	36
5.4	Number of received requests and forwarded requests per second on network $N2$ using approach PN	36
5.5	For switch SP on network $N2$, the percentage of requests dropped against the maximum instantaneous request rate for each host on network	37
5.6	For switch SH on network $N2$, the percentage of requests dropped against the maximum instantaneous request rate for each host on network	38

5.7	For switch <i>PH</i> on network <i>N2</i> , the percentage of requests dropped against the maximum instantaneous request rate for each host on network	38
5.8	For switch <i>PN</i> on network <i>N2</i> , the percentage of requests dropped against the maximum instantaneous request rate for each host on network	39
5.9	Percentage of the normal and anomalous ARP requests dropped for network <i>N2</i> with different threshold values.	40
5.10	Percentage of the normal and anomalous ARP requests dropped for network <i>N3</i> with different threshold values.	40
5.11	Percentage of the normal and anomalous ARP requests dropped for network <i>N3</i> with synthesised scans with different threshold values.	41
5.12	CDF of the number of drop requests across all three networks for various threshold values.	43
5.13	Percentage of the normal requests and retransmissions that are dropped for network <i>N2</i>	43
5.14	Percentage of the normal requests and retransmissions that are dropped for network <i>N3</i>	44
5.15	Percentage of the normal requests and retransmissions that are dropped for network <i>N3</i> with synthesised scans	44
6.1	The cost of forwarding packets using the fast path increases as a function of their size.	49
A.1	Overview of the switch design.	56

List of Tables

3.1	The number of devices and type of each of the three test networks.	15
4.1	Probabilistic dropping scheme approaches. In the big-O notation, n is the number of hosts in a network and p the number of ports on a switch. . . .	31
5.1	Network information for traces used in simulations	33

Chapter 1

Introduction

Implementing large-scale local and metropolitan area networks using Ethernet technology is becoming increasingly attractive. The self-configuring properties of such networks and the increasing capabilities of current and emerging switching devices, together with the relatively low cost of Ethernet equipment, has led to Ethernet networks growing significantly in size. With current technology it is viable to construct switched Ethernet networks that serve in the order of thousands of systems with relative ease. System administrators can continuously increase the size of their networks without investing in much engineering effort. Already, service providers have started offering metropolitan-area and wide-area Ethernet-based networks (*BellSouth Metro Ethernet*, 2006)(*Yipes*, 2006). One example of a project to create a new large scale Ethernet-based network is the 100x100 project (*The 100x100 Clean Slate Project*, 2003). The goal of that project is to provide a network that supplies bandwidth of 100 Mbps to 100 million households.

The designers of the 100x100 network opt for a clean slate approach when designing their network. One of the reasons is that they believe that traditional Ethernet networks are not scalable enough to serve a network of the required size, and their worries are shared by many researchers. One of the reasons for why researchers doubt the scalability of Ethernet networks is the use of broadcast based protocols, and their excessive resource consumption, i.e. bandwidth and processing power on network infrastructure and end-systems. However, the simplicity of broadcast based protocols, and lack of requirement for dedicated infrastructure, make such protocols very appealing. Other causes of worry include the spanning-tree construction process and the nature of address learning in Ethernet switches. Because of the increased interest in very large-scale Ethernet networks, we believe it is important to research these issues.

In this thesis we investigate the claim of limited Ethernet scalability due to the reliance on broadcast based control protocols. To that means we analyze traffic generated by a well-known broadcast based protocol on three sizeable Ethernet networks that are representative of current usage of the Ethernet technology in large domains. The protocol in question is the Address Resolution Protocol (ARP)(Plummer, 1982), used to obtain link-layer addresses for particular network layer addresses in the IP model. The results of that analysis are presented in Chapter 3.

We find that under normal operation the broadcast nature of ARP does not impose any scalability limitations on Ethernet networks. A few issues with the nature of ARP exist, that if left unhandled might be a cause for concern. However, we identify simple solutions to these problems, e.g. introducing black hole routers to networks and employing opportunistic ARP cache population. Further, the severity of these problems decreases with increased bandwidth and forwarding capacity of network infrastructure devices. We infer that in general, under normal operation, control protocols based on broadcast communication are not a scalability threat to Ethernet networks, as long as reasonable steps are taken in the protocols design and utilization.

One problem, however, is worthy of a closer look. Our analysis shows that serious performance issues can arise in the presence of misconfigured or malicious devices, such as Internet worms or viruses. More specifically, misconfigured or malicious devices can generate a large stream of ARP requests that are broadcast by every network switch and processed by every network end-system. In the case of Internet worms, infected devices perform network scans in search of vulnerable targets. These network scans result in a huge amount of ARP request traffic. As the propagation of the worm increases, so does the number of devices performing concurrent network scans. Assuming that the number of addresses on an Ethernet network is proportional to the number of vulnerable end-systems, the amount of ARP request load grows exponentially with the number of end-systems on the network. Our data shows that during the propagation of a well-known Internet worm, a large network was not operational for prolonged periods of time, due to heavy ARP request loads.

We propose a family of schemes to address the problems caused by misconfigured and malicious devices. The schemes are described in Chapter 4. The scheme limits the resources used for ARP request forwarding on a switch to a predefined threshold, defined as a portion of the switch's total forwarding capacity. The schemes drop requests exceeding the switch threshold, favoring requests broadcast by devices with high request rates. The schemes are extensions to standard Ethernet switches, and are completely local to each switch. As shown in Chapter 5, the schemes ensure that network switches do not become overloaded by ARP requests during these request storms, keeping the network operational under these extreme conditions. Further, as the schemes favor requests from well-behaving hosts for forwarding, the scheme's effect on normally operating hosts is minimal. The benefits of the using the schemes easily out-weight these minimal effects. An full blown implementation of one of the schemes on the Intel IXP1200 network processor platform suggests that the cost of running the schemes is minimal.

We will show that even problems caused by extreme behavior of broadcast based control channel protocol such as ARP can be mitigated by simple mechanisms in the network. We believe that the problem of misconfigured and malicious devices is representative to a larger set of problems associated with broadcast based protocols, and those problems can also be solved using similar approaches. We believe that both from a network theoretic and practical deployment standpoint, the cost of abandoning the current Ethernet service model, and sacrificing the simplicity of broadcast based protocols, is more than the effort required to solve scalability limiting problems caused by broadcasting protocols in a local manner in the network infrastructure.

The rest of this thesis is organized as follows. In Chapter 2 we introduce the necessary background for what follows in this thesis and what others have contributed to related topics. In Chapter 3 we present a thorough analysis of ARP behavior on three large local area networks, some of which have been the target of extensive worm propagation. In Chapter 4 we propose a family of methods to limit the effects of anomalous ARP behavior. Each family member's effectiveness and performance are evaluated in Chapter 5. In Chapter 6, we describe an implementation of the ARP management scheme on the IXP1200 network processor from Intel, and analyze and evaluate its performance. We conclude in Chapter 7. Finally, Appendix A contains a details description of the ARP managing switch design for the Intel IXP1200 network processor.

Chapter 2

Background

Inspired by the previously described attractiveness of large Ethernet-based networks, some research effort has gone into understanding their scalability. Some claim that the spanning-tree construction carried out on Ethernet switches is the main bottleneck on Ethernet scalability. This is because the extraction of a single spanning-tree from the switched mesh removes redundant links, preventing their utilization and decreasing fault-tolerance. Further, this prevents load-balancing and increases end-to-end latency as some of the shortest end-to-end paths may have included links that were removed by the spanning-tree protocol. These are well-known effects of the use of spanning-trees, and have received some research attention, e.g. in (Sharma, Gopalan, Nanda, & Chiueh, 2004)(IEEE, 2002)(Perlman, 2004)(García, Duato, & Silla, 2003). Similarly, worries of large forwarding tables due to lack of hierarchy in Ethernet addresses have been alleviated by improvements in hardware technology. However, the alleged threat of broadcast based protocols remains virtually unaddressed.

In (Myers, Ng, & Zhang, 2004), radical changes are proposed to the Ethernet service model. It is proposed that link-layer broadcast support should be removed and a directory service be used to obtain address mappings and locate services, such as DHCP. A link-state protocol is used to exchange host position information and enable unicast routing. These proposals are made on the assumption that broadcast imposes significant scalability limitations on Ethernet networks. The authors present a short survey based on ARP to support that claim. In contrast, in this thesis we show that ARP is not a scalability threat under normal circumstances, and extreme scenarios can be dealt with using simple mechanisms. We believe that it is desirable to keep native support for broadcast on Ethernets. Further, introducing a new service model would be very hard, considering the ubiquitousness of Ethernet-based networks, while our scheme could be deployed in an incremental, on-demand manner.

As described in Chapter 2.2, worms use scanning techniques to locate new targets for worm infection. Some worms scan entire subnetworks in sequential manner. Our traffic analysis shows that a large amount of ARP broadcasts are caused by the gateway trying to find the MAC addresses of unbound IP address, in response to receiving scan messages from external machines. Black hole routers/sink holes (Greene & McPherson, n.d.) are an elegant solution to deal with external scans. A black hole router is a router inside the LAN

that advertises, to the gateway, routes to the “dark IP space” (i.e. unbound/unallocated addresses and prefixes), and that drops, as well as possibly logs, any traffic it receives. The default gateway will cache the address mapping of the black hole router, eliminating the need for excessive ARP broadcasts caused by external scans. As a result, the vast majority of malicious traffic is “filtered out” of the ARP requests broadcast by the gateway (whose volume is therefore drastically reduced). Although our proposed scheme is less effective at managing (and reducing) malicious ARP broadcast from the gateway, it is easier to deploy and configure. Furthermore, black hole routers are powerless to defend against, and control, internal malicious ARP traffic emitted by local machines – a situation where our scheme excels. For that reason, our proposed scheme and black hole routers are complementary solutions.

2.1 The Address Resolution Protocol

When devices on an Ethernet network transmit an IP packet to another network device, they frame the packet in an Ethernet frame and put the receiving device’s Ethernet address in the frame’s header. The address resolution protocol (ARP) is the primary means by which network devices discover the hardware addresses associated with IP addresses for other local network devices.

2.1.1 Operation of ARP

The ARP protocol per se is address independent at both network and link-layer, it does not assume a particular link-layer protocol or network protocol. Instead, it is specialized for each protocol pair through the use of address length fields in the ARP header. We are only interested in the most common pairing, i.e. Ethernet and IPv4.

The beauty of ARP lies in its simplicity. If a device s needs an address mapping for protocol address p_t it simply broadcasts an ARP request message asking: *What device has address p_t ? Tell p_s .* This message reaches all the devices on the local area network, including the target device t . Upon receiving a request, device t will send (unicast) a reply stating, *I have address p_t and my hardware address is mac_t .* This process is depicted in Figure 2.1. In the case where a device does not receive an ARP reply following the transmission of an ARP request, the device will retransmit the request, typically up to three times.

Note that the main resource consumption of ARP takes place during the first phase of address resolution, i.e. the propagation of the ARP request. Each switch in the network has to perform expensive memory copy operations while transmitting the request on every port, and each network end-system has to use resources to receive and process the request. The ARP reply, however, is sent directly to the requesting system, and does not consume any more resources than other unicast frames. This is depicted in Figure 2.2.

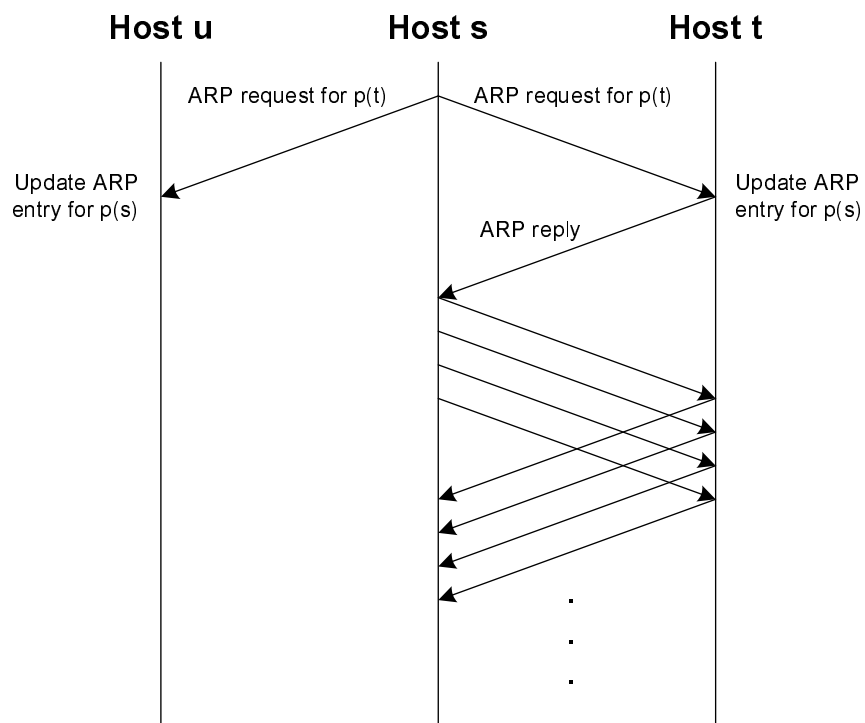


Figure 2.1: Address resolution performed by host s for the protocol address of host t . Furthermore, host u receives the request and updates its address mapping for the protocol address of host s .

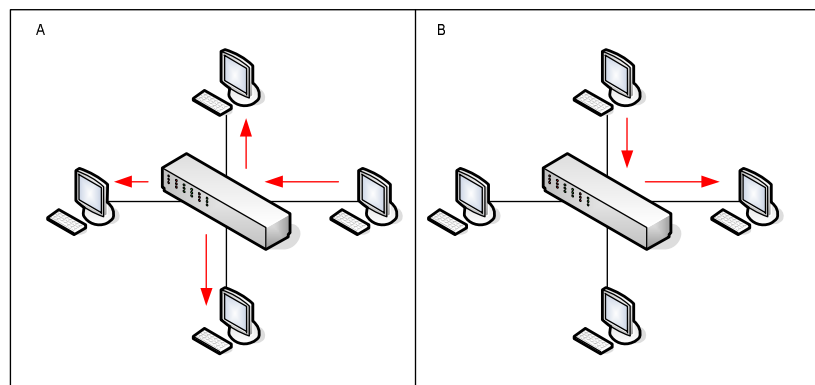


Figure 2.2: A) A host performs address resolution by broadcasting an ARP request. The switch in the center of the topology copies the frame to three ports and three end-systems process the request. B) The target replies using unicast. The switch forwards the reply frame on a single port and only the originator of the process processes the reply.

2.1.2 ARP Table

As an optimization, all devices maintain an ARP table where they cache results of address resolution requests. When an outgoing IP packet reaches the link-layer the host checks the ARP table to see if the required address mapping has been cached. Since hosts usually send a burst of packets to other hosts, as opposed to a single packet, the cache ensures that the host doesn't transmit duplicate ARP requests for every single packet in the burst. It is especially useful for network nodes to cache the addresses of frequently used nodes, e.g. the network egress points and local servers.

According to the protocol specification, devices targeted by ARP requests should store the address mappings of the requests' originators in their ARP tables. This is done on the assumption that communication is bidirectional, and eliminates the need for targets to immediately broadcast ARP requests for the originator of requests targeted at them.

To prevent outdated information from appearing in the ARP table a timer is associated with each entry in the table. Periodically, the ARP service deletes entries from the table whose timer has expired. This process is also performed when the table becomes full and new entries need to be inserted, i.e. the least recently used entries are removed to accommodate for new ones. The ARP RFC (Plummer, 1982) does not define how the aging of ARP entries should be performed or how large the ARP table should be. Therefore, these parameters vary greatly among operating system implementations.¹

2.1.3 Other ARP Uses

ARP is used in a few cases other than the normal resolution of protocol addresses to hardware addresses. Some of those will be explored now.

Gratuitous ARP

Sometimes network devices broadcast ARP requests targeted at their own protocol addresses. This is called gratuitous ARP. Gratuitous ARP is generally used to update ARP table entries located at other hosts, e.g. when a new device obtains a protocol address previously held by another device. Devices that have an ARP table entry mapping the protocol address to the old hardware address update the entry with the new hardware address.

Gratuitous ARP requests are also used by some operating system network stacks to carry out duplicate address detection. According to -microsoft support ref-, this is done as follows in Microsoft Windows XXX operating systems:

- Statically addressed computers perform a gratuitous ARP when the TCP/IP stack is initialized. Up to two more ARP requests are broadcast if no replies are received.

¹ The maximum cache size for computers running Microsoft Windows NT 4 is 512 entries and 256 entries for Linux machines. This can be compensated for by running a userspace ARP daemon.

- Dynamically configured computers (DHCP) perform a gratuitous ARP following a DHCP lease. If a reply is received a DHCPDECLINE message is sent to the DHCP server and a new address requested² (Droms, 1997).

Proxy ARP

In some cases hosts, usually routers, reply to ARP requests targeted at other devices. By this process the replying host becomes the receiver of packets destined at the device targeted by the request and accepts the responsibility of routing packets to the actual destination. This can help hosts reach hosts at other subnets without configuring routing or a default gateway (Carl-Mitchell & Quarterman, 1987).

Proxy ARP is also used in Mobile-IP. With Mobile-IP local and remote hosts are able to reach a *mobile node*, i.e. a node that has moved from its original LAN and attached to another LAN. In Mobile-IP a *home agent*, located at the original home network, acts as an ARP proxy for the mobile node. Initially, the home agent broadcasts an unsolicited ARP to update ARP table entries for the mobile node. The home agent also replies to ARP requests targeted at the mobile node. Packets destined at the mobile node are received by the home agent and dropped into a tunnel terminated at a *foreign agent*, located at the mobile node's current network. The foreign agent delivers tunneled packets to the mobile node.

Finally, proxy ARP can also be used to implement a single IP subnet across two physically separate broadcast segments. As an example, a Demilitarized Zone can be implemented on one segment and made to appear to be a part of the public segment by the use of proxy ARP.

2.1.4 Malicious Uses of ARP

As mentioned before the main advantage of ARP is its simplicity. This simplicity, however, comes at the cost of major security vulnerabilities. The address resolution protocol does not contain any authentication mechanisms, i.e. when ARP replies are received ARP daemons simply trust that they came from targeted device. Further, most implementations of ARP accept ARP replies that do not correspond to any previously sent requests.

In this subsection we will take a look at some of the major security issues associated with ARP. These include:

- Denial of Service
- Man in the Middle
- Broadcast Storms

A malicious individual can very easily carry out a denial of service attacks on local-area networks by the use of ARP. By broadcasting a single ARP reply packet on the network

² This could very easily be exploited by malicious individuals to perform DoS.

mapping an operationally important protocol address to a false hardware address a hacker can make hosts on the network use incorrect addresses when framing IP packets in link-layer frames. This results in the frames being dropped by the network infrastructure. By spoofing an ARP reply for a gateway to the Internet a hacker can cut the LAN from the Internet, a spoofed reply for a file-server cuts off data access for hosts, etc.

A closely related attack is the *man in the middle* attack. In this kind of an attack a hacker inserts himself in the path between hosts and is therefore in a position to listen to all traffic sent between these two hosts. For example malicious device M wants to intercept communication between hosts A and B . M begins by sending an ARP reply to B mapping A 's protocol address to M 's hardware address. This leads B to send frames destined at A to M . M does the same for A . M may optionally forward packets along towards A and B , making them unaware of the interception.

Finally, by broadcasting an ARP request, or replying to ARP requests, mapping IP addresses to the Ethernet broadcast address, a host can cause traffic that would otherwise be unicast to the recipient to be broadcast on the local-area network. This can lead to high traffic volumes and enable traffic sniffing. Imagine, for example, if the IP address of a network's default gateway was bound to the Ethernet broadcast address. Fortunately, this threat can be alleviated by preventing IP addresses to be bound to the Ethernet broadcast address.

These examples demonstrate how the insecurity of protocols so critical to the operation of local-area networks make the network extremely vulnerable to attacks. One bright spot is that these vulnerabilities can only be exploited by individuals with physical access to the network. Today's Ethernet-networks are mostly deployed in enterprises and institutions where users have a common interest. This may not hold on Ethernet-networks deployed in different environments, e.g. a metro-area network and other public access networks.

2.2 Internet Worms

In this section, we focus on Internet worms. As we will show, Internet worms and similar software is the single largest threat to the performance of large Ethernet networks. We will now introduce worms and their history, different kinds of worms and what we can do to protect our computer networks against Internet worms.

2.2.1 History

The term Internet worm refers to a piece of code that automatically or semi-automatically propagates itself from one machine to another. It then uses the new host as a base for further propagation. Worms are not to be confused with computer viruses. Viruses attach themselves to other executable files and rely on external mechanisms for propagation (e.g. users), whereas worms generally require no external mechanisms. The first worms³ were

³ The term "worm" itself has its origins in a science fiction story called *The Shockwave Rider* written by John Brunner in 1975.

designed in 1982 by researchers at Xerox's Palo Alto Research Center. These worms were supposed to carry out useful tasks at their host computers. Their destructive characteristics soon became apparent when a bug in one of the worms rendered the network unserviceable. The first Internet wide worm outbreak, and probably the best-known, took place on November 2nd, 1988, when Robert T. Morris released his *Internet Worm*⁴. Originally, the worm was supposed to be a proof of concept. Due to a bug in the code, however, the worm overloaded many of the systems it invaded. The worm infected several thousand machines in the first 24 hours of the outbreak.

Melissa is generally recognized as the first email worm. The worm propagates semi-automatically, i.e. it needs the recipient of the email message containing the worm to open an email attachment to trigger further propagation and any other tasks the worm may perform. Targets are generally selected from the hosts address book and message boxes. Many of the more recent worms use this form of propagation. The effects of such worms are generally limited to email servers, i.e. the network load caused by these worms is not significant.

On July 12th, 2001, the next big completely automatic worm, called *Code-Red*, started causing great problems on the Internet by attacking unpatched Microsoft IIS web-servers. The worm used the first 20 days of the month to propagate itself, followed by eight days of distributed denial-of-service attacks on the Whitehouse's web-server.⁵ The worm performed a form of scanning to locate new victims, i.e. the worm generated a random list of IP addresses and tried to invade the corresponding machines. However, the worm used a static seed for the random number generator, therefore limiting the pool of IP addresses scanned by various instances of the worm. This limited propagation considerably. A new version of the worm was released a week later, this time with a dynamically seeded random number generator. Needless to say, this approach dramatically sped up propagation.

A few days later, on August 4th, a completely unrelated worm named CodeRedII spread on the Internet. This worm used the same buffer overflow bug in IIS to install a backdoor to the system. The worm then laid dormant for 24 hours before starting probing for new victims. CodeRedII used a far more intelligent scanning technique than its predecessors, i.e. when generating random IP address to probe, it preferred IP addresses that shared a prefix with the current host's IP address. 12.5% of the time, the worm generated a completely random IP address, 50% of the time the generated address shared the first eight bits with the host's address, and 37.5% the generated address will share the first 16 bits with the host's address. This technique has been adopted by recent worms. Some worms even scan the entire local network in which they reside. The rationale for preferring systems with similar IP addresses is that these systems are likely to be maintained by the same persons, and therefore share the same security holes. Secondly, once a single instance of a worm manages to breach a network firewall, an attack from the inside is easy. A byproduct of a worm's scanning process is the generation and transmission of an excessive amount of ARP requests, i.e. each opportunistic probe performed by a worm has a very low probability of an ARP cache hit, resulting in an ARP request to be broadcast. As

⁴ Also known as the Morris Worm.

⁵ Since the worm didn't use DNS, changing the IP address of the web-server proved to be enough.

demonstrated later, this can result in a large volume of ARP broadcast traffic, especially as the number of scanning worms on a network increases.

2.2.2 Purposes and Effects

The purpose of Internet worms vary greatly. However, the techniques used to carry out a worms purpose generally fall in one of the following categories:

- Installing backdoors for future use.
- Enlisting the host in a population of zombies.
- Distributed denial of service (DDoS).
- Personal data collection.
- Propaganda.
- Spaming.

A worm may install a *backdoor* that the programmer of the worm (or anybody that is aware of the backdoor) can use in the future to gain control of the host machine. This can be useful to hide one's tracks when participating in illegal activities, e.g. hacking a third system. Another common technique is to enlist the host system in a population of *zombies*, i.e. install a backdoor-like software on the host system and report the IP address of the host to the programmer of the worm, or the owner of the zombie population. The zombies then lie asleep (or dead) until the controller signals the zombie population to perform a particular task, e.g. send out spam messages or participate in a distributed denial-of-service attacks. Often, the purpose of a worm is determined beforehand, e.g. participate in DDoS, like Code-Red, or send out spam. Other purposes of worms include the collection and reporting of personal data, e.g. credit card information and usernames and passwords, and propaganda, e.g. defacing web-sites to display propaganda, or other information the worm programmer consider humorous.

The effect of Internet worms may often be far more serious than the programmer intended. Poorly written code often unintentionally overloads or crashes the host computer, consumes excessive network resources due to aggressive probing, rendering networks unserviceable, overloads mail-servers, etc. This often limits the propagation of the worm. Worms don't generally cause any permanent harm on their host computers, whereas computer viruses' goal is often to erase or corrupt files, ruining valuable personal data.

2.2.3 Mitigation

The increased frequency of new worm spreads, and societies increased reliance on computer networks, has brought increased research focus on limiting the effects of network worms. These resulting approaches can be categorized as pro-active prevention, e.g. automatic operating system and services patch systems (e.g. Microsoft Windows Update

(*Microsoft Windows Update*, n.d.)), firewalls, etc., and reactive techniques. Reactive techniques can react in any number of ways. Further, they can reside on different places in the network, e.g. on the hosts on the network's edge, or in the network itself, i.e. on network infrastructure devices.

Snort (Roesch, 1999) and Network Security Monitor (Heberleid, Dias, Levitt K, Wood, & Wolber, 1990) are two examples of network-based worm containment approaches. The schemes presented in Chapter 4 are similar in nature to network-based worm containment approaches. While it is not a direct goal of ours to detect and contain Internet worms, our schemes could be used to reduce their rate of proliferation. As such, we believe the our schemes are complimentary to these approaches.

Virus Throttling ((Williamson, 2002)(Twycross & Williamson, 2003)) is an end-system based system designed to slow down the propagation of Internet worms. The system is an extension to the end-system network stack, and works by limiting the rate of new connections initiated by the host machine. The fact that the system relies on end-systems means deployment is difficult. Additionally, it is not clear that malicious software cannot bypass the mechanism and transmit messages directly to the network.

Chapter 3

Network Analysis

I never could make out what those damn dots meant.

- Lord Randolph Churchill, 1849-1895

To gain an increased understanding of ARP behaviour on large local area networks, we carried out extensive measurements and data analysis of ARP traffic on three local area networks. The data included ARP request traffic collected across three relatively large networks that serve our respective institutions and a student residential network. The data included periods of extensive ARP flooding where several network devices infected by an Internet worm (Symantec, n.d.) scanned the local network for machines vulnerable to infection.

Network	#Devices	Type
N1	ca. 700	University Network - Computer Facilities, Private Workstations
N2	ca. 2900	Residential Area Network - Private Workstations
N3	ca. 3800	University Network - Computer Facilities, Private Workstations

Table 3.1: The number of devices and type of each of the three test networks.

The three networks (summarized in Table 3.1) are all similar in structure. The smallest network (*N1*) consists of approximately 700 end systems, most of which are workstations of students and faculty members. Additionally, the network contains gateways to the Internet, a wireless local area network, and local research laboratories. Finally, the network contains local servers, such as storage, web, domain, print, name, and DHCP servers. The second network (*N2*) is a student residential network. It contains about 2900 hosts with a single gateway to the Internet and a few local servers. The majority of the machines are the private workstations of residents. The third network (*N3*) consists of approximately 3800 hosts interconnected by a 100 Mbps switched network. One egress point to the Internet exists on the network, along with a number of devices providing key services such as those on *N1*.

The broadcast nature of ARP requests make the collection of ARP traffic data on a network trivial. This was done on a single machine attached to each network, using *tcpdump* to store the ARP requests in files. The ARP replies, however, are delivered using unicast, therefore making their collection very difficult, due to the switched nature of modern Ethernet networks. Consequently, our data only includes the ARP requests. This does not limit our measurements and analysis since the unicast nature of replies is unlikely to impose significant overheads on Ethernet networks. To aid our analysis, a set of tools have been written to extract and process the collected ARP request traffic.

In the following sections, we present the results of this analysis. The aim is to identify what constitutes normal ARP behavior. With an understanding of normal ARP behavior, we will point out and quantify the anomalous behavior that is caused by misconfigured and malicious devices.

3.1 ARP Broadcast Distribution

The data presented in Figure 3.1 shows that the number of devices broadcasting ARP requests at any given moment is fairly low. There are, however, fairly frequent spikes where a substantially larger number of devices flood the entire network with mapping requests. We also note that ARP request spikes observed on *N1* are as high as one third of the number of nodes on the network.

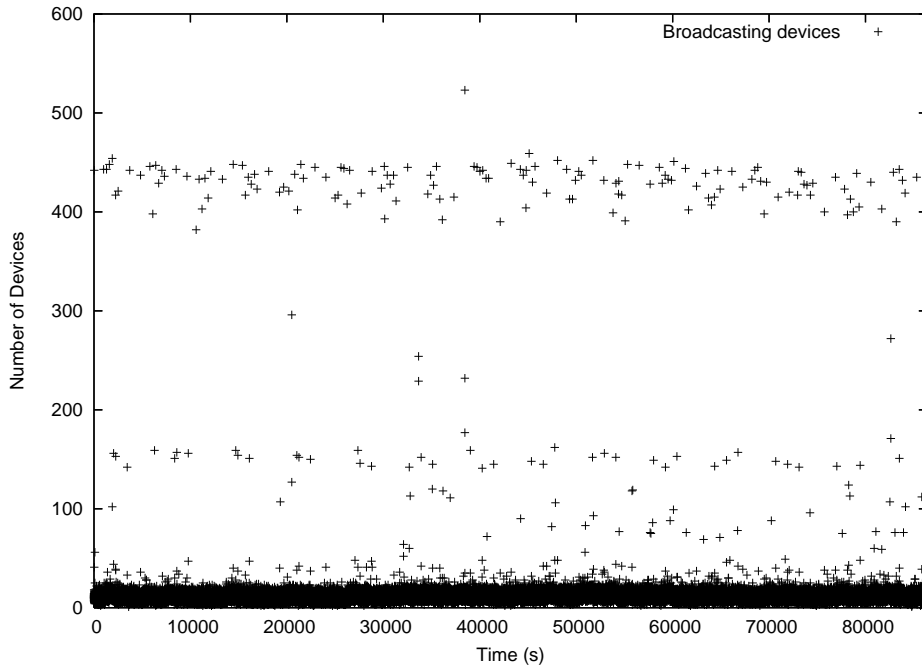


Figure 3.1: The number of devices broadcasting ARP requests at a given second over one day, taken from *N3*

These spikes were found to correspond to events where a large set of devices broadcast ARP requests for a single device. A closer look at a short period before and during the spikes reveals that almost every host that broadcasts requests during the peak times lacks

a mapping for the same IP address. This kind of behavior can only be caused by an event triggered by the end system holding the IP address targeted by the wave of ARP requests. A look at other (non-ARP) broadcast traffic reveals that the target of the ARP requests had broadcast a higher-layer protocol message, such as a NetBIOS name query, that required a subset of the network's devices to respond. The respondents, lacking an address mapping for the originator, each broadcast an ARP request.

Figure 3.2 displays the number of ARP requests broadcast on the *N3* for a single day. There are a lot of spikes, some of which correspond to the spikes mentioned earlier. Others spikes correspond to a single or a small subset of devices broadcasting a large number of requests in a short period of time.

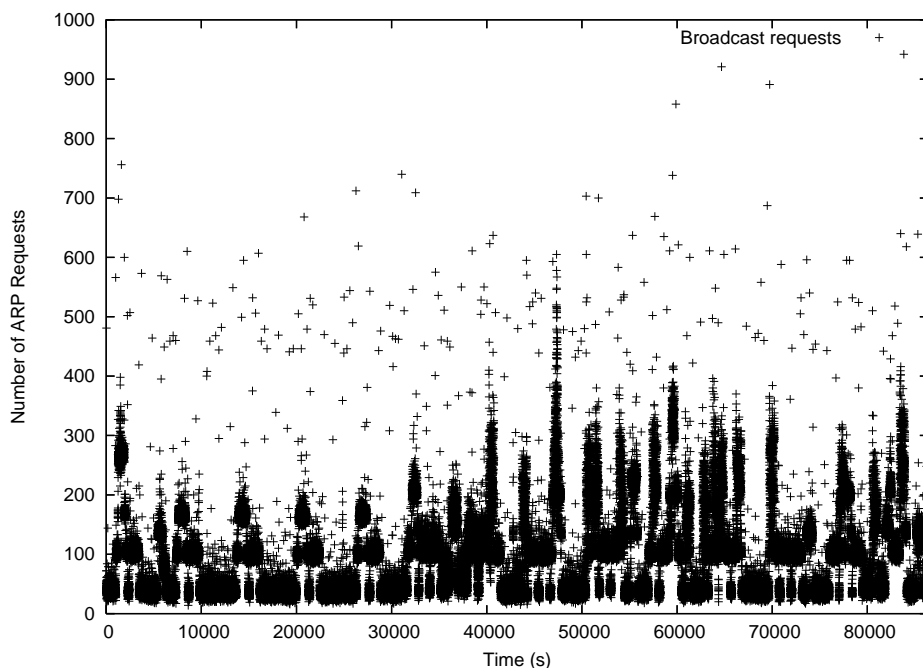


Figure 3.2: The number of ARP request broadcast at a given second over one day, taken from *N3*

The latter class of spikes can be explained by two forms of behavior. The first form of behavior relates to servers that keep a list of their clients. The clients are contacted, either periodically or as an effect of a higher layer event, and triggers the need for an address mapping. If the interval between these contacts is larger than the ARP table timeout, or the number of clients is larger than the ARP table size, this results in an ARP request for each client.

The second form of behavior observed that causes these spikes, relates to devices scanning the network, thus triggering ARP requests for every address in the network's address range. One of our data traces was taken at a network (*N2*) heavily infected with scanning worms. Scanning can also be performed by an external node, in which case for example, the external node issues an ICMP echo request for IP addresses on the network¹. In the latter case, the gateway router of the network is the initiator of the ARP requests. Scanning is generally performed by malicious software, for example Internet worms (Weaver,

¹ This only applies to networks where the hosts have public IP addresses, i.e. not behind NAT.

Paxson, Staniford, & Cunningham, 2003) looking for machines vulnerable to infection (see Chapter 3.2). Today, no attempt is made to limit the effects scanning worms have on the performance of Ethernet networks.

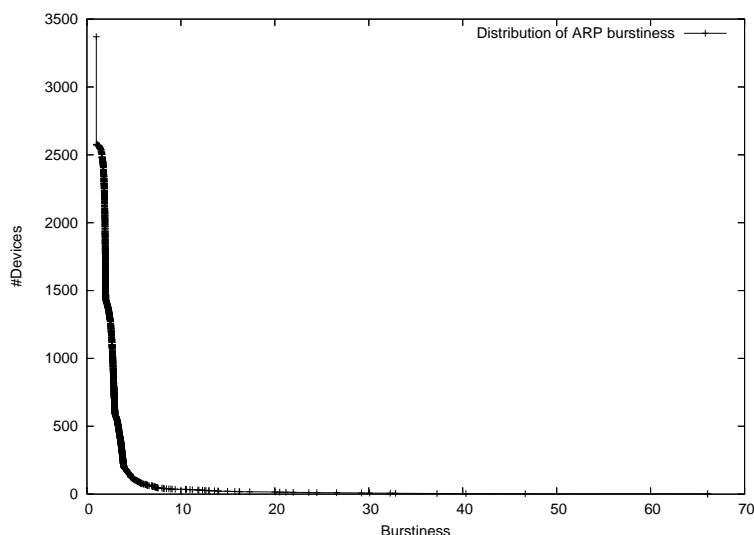


Figure 3.3: Burstiness characteristics of ARP traffic

An interesting characteristic of ARP request broadcast per device is its burstiness. If we define the maximum instantaneous ARP request rate for a device to be the inverse of the shortest observed inter-request time between two consecutive requests from that device, and the average ARP request rate to be the overall number of requests divided by the length of time over which these were observed, we can then define burstiness as the ratio of the maximum request rate to the average rate. Figure 3.3 plots the number of devices whose burstiness is greater than the corresponding value along the x-axis for one of our test networks, but is representative of all our observations.

In Figure 3.3, we see that most devices in normal operation do not send ARP request broadcasts in bursts. In other words, it is not normal for a device to broadcast a substantially larger number of requests over a short period than on average. Small bursts may occur when hosts start up and duplicate address detection (described in Chapter 3.2) is performed, and the host acquires the address mappings for critical services such as DNS servers and gateways.

A key observation is that the traffic volume generated by ARP requests is very low on average. On the largest studied network (*N3*), the average number of broadcast ARP requests per second is 85, which corresponds to approximately 43 kbps. If we assume linear growth, a network of 85000 devices would generate a volume of about 1 Mbps: 1% of the capacity of standard fast Ethernet networks today. During the dissemination of an Internet worm, however, the volume increases dramatically.

3.2 ARP Traffic Patterns

The ARP data analysis reveals an interesting characteristic of the hosts on the network: their traffic patterns. The data shows that the ARP requests broadcast are almost symmetric. In other words, a small subset of devices are targeted by a large subset and, likewise, the few devices broadcast many requests for the large subset. Expectedly, it turns out that the few heavily hit devices are the network's gateway routers and servers. Almost every single workstation needs to communicate with the local servers and with non-local servers/peers through the gateway. Therefore, a lot of ARP requests broadcast by workstations target these devices.

Perhaps more surprisingly, a lot of requests broadcast by the servers and gateways were observed that targeted the local workstations. In the initial address mapping acquirement phase, performed when workstations boot and start communicating with other network devices, the workstations learn the hardware addresses of the gateways and servers. Since the targets of ARP requests should add the source of the request to their ARP tables, a single ARP request should suffice for each pair of communicating devices. From that point forward, the ARP table entries at the two devices should be refreshed at roughly the same time, assuming communication is generally bidirectional (e.g. data and acknowledgements in a TCP connection). The fact that servers and gateways frequently request address mappings for workstations does not conform to this line of reasoning.

One reason for the large number of requests broadcast by servers and gateways seems to be caused by the fact that the ARP table on these devices is not large enough to contain the entire set of active clients, i.e. address mappings are removed from the ARP table prematurely to accommodate for new ARP table entries. Shortly after prematurely removing an address mapping from the table, a device broadcasts an ARP request for the recently removed address mapping, removing another entry prematurely when storing the result in the already full ARP table and triggering yet another ARP request. This chain reaction greatly reduces the effectiveness of the ARP cache.

Another reason for high request rates of gateways is externally sourced address-range scans. When external network devices perform scans, packets enter the local area network through a network's gateway. At that point the gateway needs to map the destination IP address to its associated link-layer address. This causes the gateway to check its ARP table for an entry containing the correct address mapping. If no entry is found, the gateway broadcasts an ARP request packet.

Since the device performing the scan does not know what IP addresses have been allocated to devices on the network, many of the probe packets will be destined to unbound addresses. Each packet will trigger an ARP request which will not yield a reply. The data from *N3* shows that about 81% of requests sent by its gateway are for unbound addresses, which suggests a significant number of external scans.

Finally, some operating systems implement the ARP timeout mechanism such that it verifies the validity of timed-out entries, instead of just removing them. This is done directly by using *unicasting* and ARP request. If the target doesn't respond ARP *broadcasts* an ARP request for the IP address, potentially discovering a new address mapping for

the target address. Since the set of hosts is more dynamic than the set of gateways and servers, and communication tends to be between devices in these sets, rather than within the sets, ARP requests due to address mapping verification are mainly sent by gateways and servers.

Figure 3.4 highlights an interesting application of ARP. Some network devices broadcast ARP requests targeted at the protocol address they currently hold. This is called gratuitous ARP and can be seen as a diagonal line on the figure. Some operating system network stacks use self-targeted ARP requests to perform duplicate address detection. This behavior is not of significant concern in the context of the scalability of ARP, since the number of requests generated is of $O(n)$ and only occurs when hosts boot.

Finally, workstations do not communicate directly with each other very often. Most communication seems to be originated by workstations and destined for local and external servers.

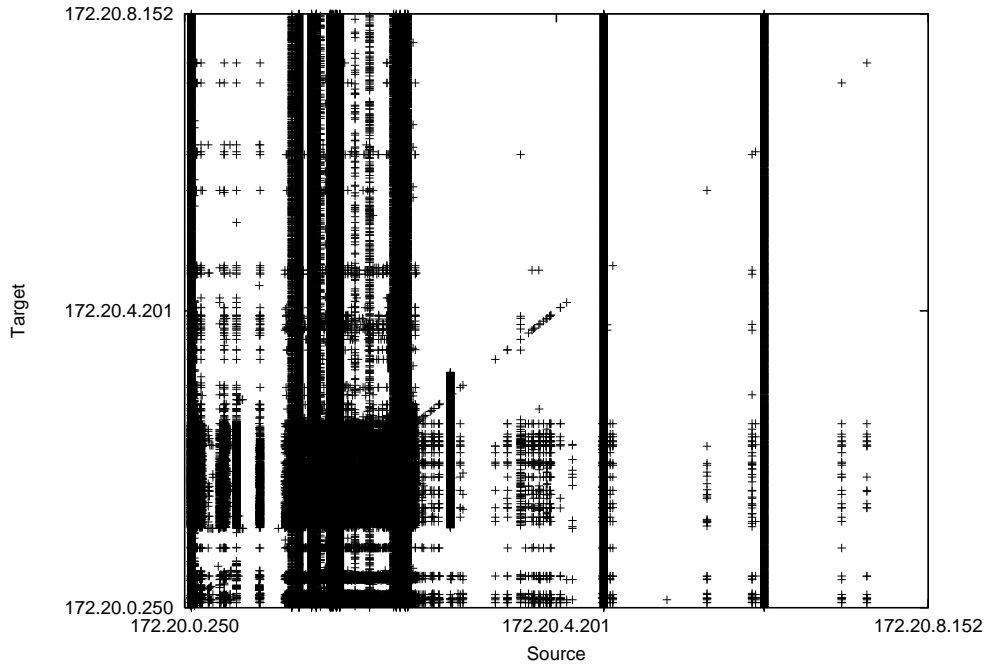


Figure 3.4: Traffic patterns showing the relationship between ARP sources and targets. This graph plots data from network *N1* with the axes representing device identifiers.

3.3 The Effect of Malicious and Misconfigured Devices

As mentioned earlier, one of the data sets used in our ARP analysis was taken during a period when a number of local devices were infected with a scanning Internet worm. Although the worm infection on that network reached its heights several weeks earlier and most of the devices had been patched, the data still contains some useful information about the ARP behavior of malicious software. This particular worm scans the local subnet for vulnerable machines in a sequential order before probing for vulnerable non-local machines. Some other worms may use selective random scan, i.e. probe targets are

chosen at random with a bias for local machines. The reason for the complete sequential scan or the bias for local addresses is twofold. First a single firewall breach allows a worm to infect the whole protected subnet. Second, machines on a local area network are likely to be managed by the same individuals and be fairly homogeneous, therefore sharing the same vulnerabilities.

Each local probe by a worm results in an ARP request broadcast by the infected machine. For the worm to speed up its dissemination, it aims to probe as many potential victims as possible in the shortest period of time. This results in a sudden increase in ARP requests rates on infected machines. When the student residential network ($N2$) got infected by the Internet worm, several of the devices connected to the network began scanning for vulnerable machines to infect. These devices, previously broadcasting only the occasional ARP request, suddenly started to broadcast at 120 requests/second. This behavior is demonstrated in Figure 3.5. One particular day saw about 120 separate devices performing scans repeatedly. It should be noted that the data were taken several weeks after the time of the initial worm dissemination when the network became saturated by ARP scans.

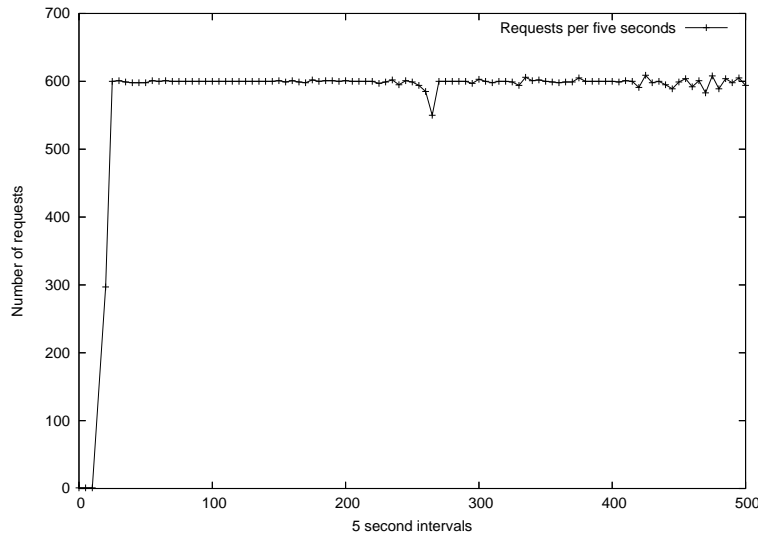


Figure 3.5: The increase in request rate for hosts infected with a worm scanning the local network for vulnerable machines.

To get an idea of how worms effect the performance of Ethernet LANs it is useful to get a sense of the dissemination of worms. In (Chen, Gao, & Kwiat, 2003) a model for calculating the spread of worms is put forward. We have extended that model so that devices stop scanning after sending a probe to every address on the local area network. The equations for the spread of scanning worms are:

$$\begin{aligned}
 m_i &= \sum_{j=0}^i n_j & s_{i+1} &= \sum_{k=i-\frac{T}{s}}^i n_k \\
 n_{i+1} &= [N - m_i] \left[1 - \left(1 - \frac{1}{T} \right)^{rs_{i+1}} \right]
 \end{aligned} \tag{3.1}$$

where N is the total number of vulnerable devices, T is the size of the address range from which probe targets are chosen at random, r is the scan rate. m_i , s_i and n_i represents the

number of devices infected, the number of scanning worm instances and the number of new infections at the end of time slot i , respectively. We ignore the death and patch rates of vulnerable devices.

A hypothetical example, based on what we have seen at the residential network, assumes a single initial infection ($n_0 = 0$), a B-address local network ($T = 2^{16}$) and a scan rate of 120 request/s ($r = 120$). The network has about 3000 devices, 10% of which are assumed to be vulnerable ($N = 300$). Figure 3.6 shows the dissemination and ARP request volume for the first 40 seconds. The spread is alarmingly fast; the worm reaches complete infection in about 25 seconds. Interestingly, and perhaps counter-intuitively, a smaller vulnerable set slows down the infection rate. If we have 30 vulnerable devices, other things being equal, it takes the worm about 140 seconds to reach complete infection. This is due to the fact that the larger the vulnerable set the more likely a probe will hit a vulnerable device. The infected device immediately starts scanning, increasing the rate of infection. This perhaps promotes the case for smaller subnets.

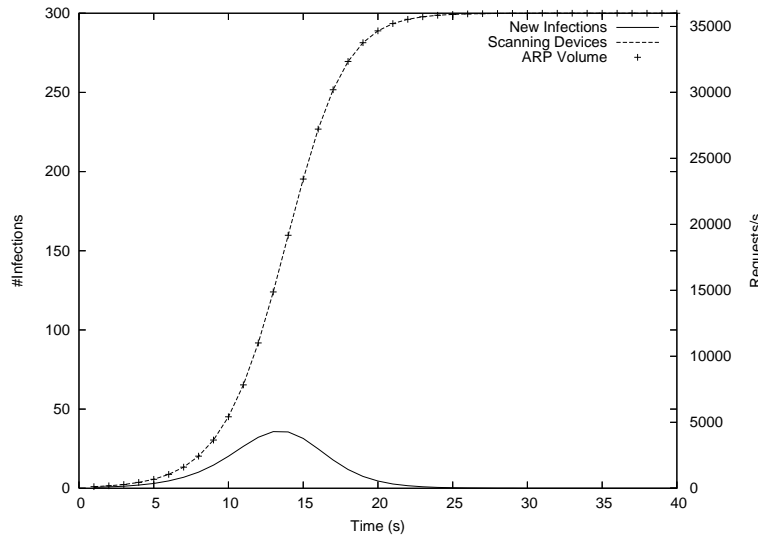


Figure 3.6: The infection dissemination and ARP volume caused by infected devices for $0 \leq i \leq 40$

ARP requests are minimum-sized Ethernet frames of 64 bytes. As these ARP requests need to be broadcast on the network, a n -port switch must replicate these on $n - 1$ of its output ports. Most switches use a slow path for broadcast forwarding where the CPU has to copy the frame to all output ports. This will reduce the performance of many switches dramatically. On a network with m infected devices, each of which broadcasts r probes/s, the switch has to forward $O(mnr)$ requests. Consequently, a few infected hosts can quickly consume a large amount of the forwarding capacity of a switch. In the above example we have about 300 devices broadcasting 120 request/s each. A 24 port switch would have to serve about 828.000 requests/s using the slow path.

Even more serious, though, is the effect it has on the performance of end-stations, each of which has to serve an interrupt, copy the packet to main memory and have a look at its internals. DMA is rarely used for packets this small. A ball-park estimate of $6 \mu\text{s}$ per interrupt, context switch and packet copy, and 36.000 requests/s results in a workstation

wasting about 22% of its CPU cycles². Interrupt coalescing, were implemented, will reduce the interrupt overhead somewhat, though. Additionally, the requests take up some precious bandwidth on downlinks to end-stations, or about 18% of 100Mbps links.

Unusually high ARP request rates are an identifying characteristic of machines infected with Internet worms. As seen earlier in this analysis, it is also typical for gateway routers to exhibit higher request rates than normal hosts. This point highlights the difficulty in distinguishing between non-malicious devices such as gateway routers and those infected with Internet worms, for example. Perhaps a more effective approach for detecting Internet worms may be to monitor the relative increase in rates at a given host.

Another characteristic that is typical for machines infected with a worm is the ratio of the requests targeted at a host to the requests broadcast by the host. This ratio is very low for machines performing address range scans while it is close to one for other hosts. This can be seen in Figure 3.4, where the vertical lines representing scans do not have corresponding horizontal lines, representing requests targeted at the scanning machines.

The number of broadcast requests targeted at unbound IP addresses is also an identifying characteristic of devices infected with malicious software. Malicious software, not knowing what IP addresses are in use on the local network, broadcast requests blindly in a sequential or random manner. On a network where address utilization is low, only a small set of requests result in a reply. This behavior is also typical of incorrectly configured devices. During the time of our observations, one of the test networks contained an incorrectly configured print server that constantly broadcast ARP requests for an unbound IP address that used to belong to a printer.

² The numbers are loosely based on (Zec, Mikuc, & Zagar, 2002) and analysis of Linux data path performance.

Chapter 4

Managing Anomalous Behavior

There is a fine line between helping administrators protect their systems and providing a cookbook for bad guys.

- Grampp and Morris, "UNIX Operating System Security"

As seen in Chapter 3, a small subset of devices on a local area network that are either infected with malicious software or incorrectly configured, may generate bursts or consistently large amounts of ARP request frames. If left unmanaged, this behavior could cause an unacceptably significant quantity of network resources to be consumed that could cause a network to become unserviceable. In this section, we describe a strategy that can be used to ensure such devices do not consume an unacceptable quantity of network resources due to this behavior.

When designing the ARP managing mechanism, we had multiple design requirements in mind. Different design decisions/options were evaluated against those requirements. We now list those requirements, roughly from the most important to the least important.

The basic requirement is that the scheme should prevent anomalous ARP traffic from rendering the network unserviceable. Further, to ease deployment, the mechanism should be completely transparent to end-system devices and other network devices. As an optimization on the basic design requirement, the effect malicious and/or misconfigured software has on behaving hosts should be limited. As resources are scarce, and performance is a huge issue when designing critical network components, the scheme's resource consumption should be kept to a minimum. This applies to processing power, memory, and bandwidth. To limit the managerial overhead, and ease deployment, the mechanism should be completely autonomous and self-configuring. Yet, if possible, the scheme should be able to be as flexible as possible, e.g. be aware of network semantics when managing ARP request traffic. Finally, as a low priority optimization, the scheme should minimize the resources consumed by anomalous behavior, hence limiting the spreading of Internet worms. This prevents other hosts from being infected, limiting future ARP request traffic.

So that anomalous ARP request traffic cannot cause a degradation of service on a network, we extend the functionality of networked switches. In short, if an observed ARP request rate exceeds a threshold (specified in requests per second, for example), we invoke an algorithm that monitors the rate of ARP request streams and performs probabilistic drop-

ping of requests based upon the observed stream rates. The probabilistic dropping mechanism is designed such that the streams with higher rates have a greater probability of their requests being dropped. This scheme continues until the ARP request rate falls below the threshold, when the scheme is reset and “unmanaged” ARP forwarding resumes.

The observed rate of an ARP stream can be monitored from a number of different *scopes* – for example, host, port, switch, or network. The granularity of a scope determines to what extent devices effect each other. Therefore the finer the granularity of a scope, the fewer devices affect each other. For example, in Figure 4.1, if using a host scope, host A does not affect any other hosts. However, if port or switch scopes are used, the number of hosts affected by host A increases. When a scope’s threshold is reached, the probabilistic dropping algorithm is run for that scope only.

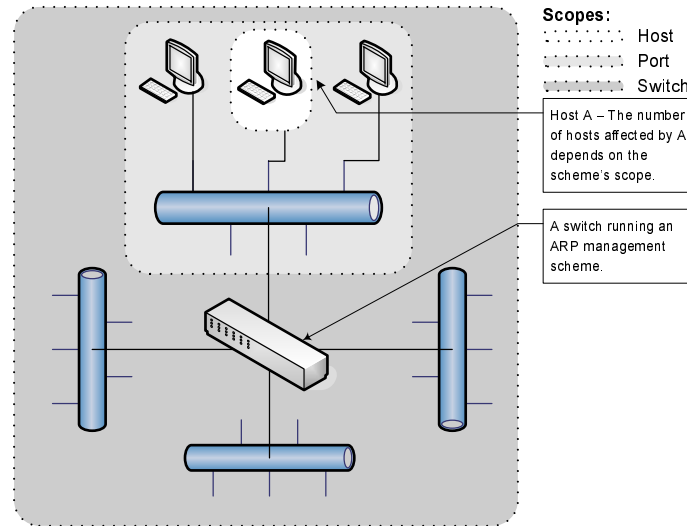


Figure 4.1: Different scheme scopes. The number of end-systems that affect each other is dependent on the size of the scope.

There are a number of factors that influence the most suitable way to identify ARP streams. In our scheme, ARP streams are identified either by the switch port or host from which they originate. When identifying ARP streams by the port on which they are received, the observed rate is the aggregate of the request rates of all hosts connected to that port (directly or through another switch/hub).

4.1 Probabilistic Dropping Scheme

We now go on to describe the general probabilistic ARP request dropping mechanism. A networked switch defines a threshold for each of its scopes. If the observed rate of received requests exceeds the threshold, the dropping mechanism is engaged for the corresponding scope. Each ARP stream (identified by a stream identifier as described above) observed in the scope is allocated a share of the available resources allowed for request forwarding (the threshold). If a stream exceeds its share, ARP requests belonging to the stream are probabilistically dropped. This dropping is carried out in such a way that

streams that attempt to consume larger amounts of a scope's resources will suffer an increased probability of having their requests dropped. The mechanism runs until the aggregated ARP request rate in the scope falls below the threshold, at which point the "scheme" is reset and normal ARP forwarding resumes.

Data: v - a switch wide moving average request rate. Initialized to 0.

Data: $table$ - a table containing devices seen and timestamp of last request seen.

```

1 begin
2   foreach incoming requests req do
3      $s \leftarrow \frac{1}{\text{now} - \text{last request arrival}}$ 
4      $v \leftarrow (1 - \alpha) \times v + \alpha \times s$ 
5     if  $v \geq \text{threshold}$  then
6        $\text{add}(table, req)$ 
7        $f = \frac{\text{threshold}}{\text{size-of}(table)}$ 
8       if  $\text{allow-request}(req, f)$  then  $\text{forward-request}(req)$ 
9     end
10    else  $\text{forward-request}(req)$ 
11  end
12 end

```

Algorithm 1: Control ARP

input : Request req and the fair share f .

output: Boolean indicating if the request should be dropped or forwarded.

```

1 begin
2    $l = \text{get-update-timestamp}(table, \text{sender-of}(req))$ 
3    $r = \frac{1}{\text{now} - l}$ 
4    $p = \max(0, \frac{r - f}{r})$ 
5   return  $\text{throw-coin}(p)$ 
6 end

```

Algorithm 2: Allow Request

To be more specific, the probability p_i that a stream i will have an ARP request dropped becomes higher as the request rate for i (r_i) increases relative to its share of the resources allocated to ARP request forwarding (f). This relationship is defined in Equation 4.1, where the value r_i is the request rate for host i . The rate is calculated in different manners depending on the type of ARP stream identifiers used (see Section 4.3). Here, l_i represents the time at which the previous request of stream i was received. The rate and drop probability are calculated in Algorithm 2.

$$p_i = \max(0, \frac{r_i - f_i}{r_i}) \quad (4.1)$$

$$f = \frac{t}{n} \quad (4.2)$$

$$v = (1 - \alpha) \times v + \alpha \times s, \quad \text{with } 1 \geq \alpha \geq 0 \quad (4.3)$$

$$s = \frac{1}{now - \max_i l_i} \quad (4.4)$$

As mentioned earlier, the f value represents the share of the scope's ARP request forwarding resources that are allocated to an individual request stream. This value is determined using Equation 4.2, where the value t (given in requests/sec) represents the scope's maximum desired ARP request forwarding rate (threshold), and n is the number of streams observed in the scope since the dropping scheme was invoked.

The streams known to the dropping scheme are forgotten and not used to calculate the share of the scope's available resources after a timeout period. This is so that the share of the available resources (f) is not effected by streams that have not been active for extended periods. A separate maintenance algorithm removes stale entries, such that they are removed when $now - l_i > D$, where D is the timeout value for device entries. Forgetting of streams in this way has two consequences – one is to decrease the likelihood of dropping requests belonging to streams with low request rates, and the other is to increase the fair share f , which allows malicious software to send at a higher rate. This is only a minor problem as the aim of the scheme is to limit the resources used on ARP request broadcasting to t , therefore limiting the effects of malicious and malfunctioning devices, while limiting the resource consumption of individual devices or slowing down worm propagation is a low priority design requirement.

The observed rate in a scope (v) is computed as a moving weighted average of request rate samples, as in Equation 4.3, because the combination of non-bursty ARP traffic streams from hosts can result in some burstiness in a scope. This prevents the dropping scheme from starting up unnecessarily in case of transient spikes, i.e., the scheme only starts dropping requests when a high request rate has been experienced for a sustained period of time. This average is calculated in Algorithm 1.

4.2 Scopes

As demonstrated before, the dropping scheme can run in any of multiple different scopes (host, port, switch, network). Each of the scopes has its advantages and disadvantages. The two main approaches we have used are the port and switch scopes.

In a switch scope, a single threshold value is defined for the entire switch and a single switch-wide average request rate is monitored. If the observed request rate exceeds the switch's threshold the dropping scheme is engaged and all ARP requests received are subject to probabilistic dropping. A request's drop probability is equal to its associated stream's p_i value as described in Equation 4.1. This approach has the advantage that its computational resource consumption is relatively limited compared to the other scopes, i.e. a single threshold is defined and a single average rate is calculated for the entire switch. However, it has some potential drawbacks. Because the scope is unnecessarily wide, once the scheme is invoked, requests generated by a larger number of hosts than strictly necessary are subject to dropping. Further, rate state is maintained for every observed stream on a switch, increasing memory requirements. As a large threshold is

needed to accommodate for the ARP request traffic generated by every network device, malicious software has increased room to misbehave. Finally, a switch scope scheme is totally unaware of network semantics and its flexibility limited, i.e. different devices cannot be allowed to send larger volumes of ARP requests.

In a port scope, a threshold value is defined for each port on a network switch. Further, the average request rate is monitored at each port. If the observed request rate reaches a port's threshold the scheme is engaged on that port only and only the ARP requests that are received at that port are subject to probabilistic dropping. This approach eliminates many of the shortcomings of the switch scope approach. As the granularity of a port scope is finer than that of a switch scope, the number of normally operating hosts affected by a single (or few) anomalous streams is reduced. For the same reason, the state maintained by the scheme while it is engaged is significantly smaller than in a switch scope. The thresholds for port scopes can be defined as a portion of a switch-wide threshold, or otherwise configured. The only requirement is that the sum of the port thresholds does not exceed the switch's forwarding capacity. Thus, a port scope gives anomalous streams less room to affect other devices, and increases the schemes flexibility. The only disadvantage of the port scope approach is that the implementation complexity is increased, especially with regard to threshold value definition.

We could have also considered host and network scopes. When a scheme's scope is individual hosts, the scheme has to constantly monitor the ARP request rates of all observed hosts. This has the advantage of being unforgiving to hosts running malicious or misconfigured software. Different devices, however, have different ARP forwarding needs, which makes the division of a switch's total ARP forwarding capacity among all scopes running on the switch difficult. Further, the memory and state maintenance requirements of such a scope design are excessive and the calculations are more complex. Finally, this design would mean that the switch was constantly monitoring the ARP request rates of every host on the network. This breaks our requirement that the processing power resource consumption be kept minimal. Instead, the scheme should only run occasionally to limit the effects of malicious or misconfigured software, such as when a new Internet worm outbreak takes place, and otherwise leave the switch to carry on with its usual responsibilities. A scheme having a scope of the entire local area network would require a complex distributed algorithm and causes a considerable communication overhead. As stated before, one of our design requirements was that the mechanism would be fully autonomous, robust, self-contained, and transparent to the networks end-system devices. This effectively rules out a network-wide coordination of switches to manage ARP traffic.

4.3 Stream Identifiers

ARP request streams can be identified using one of two methods, i.e. using the source address of received requests, or the port number on which requests are received.

When identifying streams on a per-host basis, the current request rate has to be maintained for each host that is transmitting ARP requests when the dropping scheme is invoked.

This state needs to be maintained only on a per-port basis if streams are identified by the incoming port number. When per-port identification is used, well behaved hosts may be penalized if they share the same port as a misbehaving host (i.e., one with a particularly high request rate). Clearly, if a host is connected directly to a switch's port, as is becoming the most widespread scenario, the two identification schemes are equivalent.

When using the per-host identification scheme, a problem could occur if a misbehaving host masquerades the MAC address of another host. A malicious host could generate spurious ARP requests pertaining to another host, which could be used as a DOS attack. The effect is a reduced fair share for all hosts in the corresponding scope (caused by an increased number of observed hosts), except for the malicious host which would receive an increased fair share (multiple slices of the total threshold).

The method used to calculate the observed request rates (r_i) are dependent upon what stream identifiers are used. In the case of per-host identifiers, the request rate is the instantaneous rate of received ARP requests (Equation 4.5) while a scheme using per-port identifiers uses a weighted moving average request rate (Equation 4.6). The reason for this is that while the requests generated by a well-behaved host are not bursty (see Chapter 3), transient bursts can occur for multiple hosts connected to the same port.

$$r_i = \frac{1}{now - l_i} \quad (4.5)$$

$$r_i = (1 - \alpha) \times r_i + \alpha \times \frac{1}{now - l_i}, \quad with \quad 1 \geq \alpha \geq 0 \quad (4.6)$$

Identifying streams based on their source hosts is applicable in scenarios where a large number of hosts are connected through one or more of a switch's ports. This translates to the internal switches of a hierarchical topology. Per-port identifiers, however, are applicable if only a single or few hosts are connected to each port, i.e. at the edge of a network.

When used in combination with a switch scope, not identifying streams is not an option unless resources (e.g. memory) are particularly scarce, as this causes an unacceptable amount of legitimate ARP requests to be dropped. It should be noted, though, that this would still be better than not managing ARP request traffic at all, since resources allocated to ARP broadcasting are limited, preventing the network from becoming totally unserviceable for nodes sharing the scope of the anomalous ARP request stream sources. Omitting stream identification is attractive in some scenarios though, especially when used in port scopes on devices deployed on the edge of the network.

4.4 Permutations

We now go on to list the possible permutations of scheme scopes and stream identifiers to introduce the different approaches of managing anomalous ARP behavior. The approaches all share the common general dropping scheme described in subsection 4.1. Each of the approaches have different strengths and weaknesses, require different amounts of resources, are unequally equipped to discriminate between legitimate and anomalous

ARP requests, and are applicable in different scenarios. The proposed approaches are listed in Table 4.1. The table clearly shows the tradeoff between complexity and resource consumption on one hand and accuracy and effectiveness on the other.

Acronym	Scope	Str. IDs	Mem	Complexity	Accuracy	Effectiveness
<i>SN</i>	Switch	N/A	$O(1)$	*	*	*
<i>SP</i>	Switch	Port	$O(p)$	* *	* *	* *
<i>SH</i>	Switch	Host	$O(n)$	* *	* * * *	* * * *
<i>PN</i>	Port	N/A	$O(p)$	* * *	* *	* * * *
<i>PH</i>	Port	Host	$O(\frac{n}{p})$	* * * *	* * * * *	* * * * *

Table 4.1: Probabilistic dropping scheme approaches. In the big-O notation, n is the number of hosts in a network and p the number of ports on a switch.

The first approach (*SN*) is the simplest. It also drops ARP requests in the least intelligent manner of all the approaches. As mentioned in subsection 4.3, this scheme is not a viable option except in severely constrained environments. The combination of a switch scope and a per-port stream identifiers, however, also requires very little resources and does a much better job of dropping requests belonging to anomalous ARP streams in favor of those belonging to normal streams. The reason is that requests arriving on ports with request rates lower than the port’s fair share are not subject to being dropped. This approach is most effectively deployed on the edge of networks, i.e. where only a single host is connected to each of the switch’s port. Even in this scenario, however, the approach requires the switch’s ARP forwarding resources to be allocated to ports in an intelligent manner, so that hosts connected through the uplink port will not have a significant number of their messages dropped. This will be discussed further in the Discussion section at the end of the paper.

A scheme that uses a switch scope in combination with per-host stream identifiers (scheme *SH*) will require more resources, as the instantaneous request rate has to be stored for each host. The scheme will do a good job of dropping anomalous ARP traffic in favor of normal traffic. It is, however, vulnerable to the security attacks described in the 4.3. It is most effectively deployed at the center switch of a star-like network topology.

A scheme that uses a port scope and drops ARP requests regardless of the stream identifiers (*PN*) is similar to scheme *SP*. However, the scheme is more effective, because the port thresholds are generally smaller than switch-wide thresholds. The scheme is applicable in similar situations as *SP*, i.e. on the network’s edge. Perhaps the most accurate, effective, and versatile approach is the combination of port scopes and per-host stream identifiers (*PH*). The state maintained by this scheme is less than in the switch-scoped approach, because the number of hosts sharing a port is smaller than the number of hosts sharing a switch (i.e. all hosts in a network). The port scope approach makes the scheme effective, as it starts dropping traffic sooner than a switch-scoped approach. Further, the combination of port scopes and per-host stream identifiers makes the dropping accuracy optimal among all the approaches. Like *SH* the scheme can be deployed in a star-topology’s center. Finally, it can be deployed on the edge like *SP* and *PN*. For ports connected to a single host, the accuracy of the switch becomes the same as in those approaches. However, the accuracy for the uplink port is increased significantly because

of this approach's ability to discriminate between different hosts. The same is true if the presence of layer 1 hubs prevents the deployment of an ARP managing scheme directly on the edge. The state maintained in this scheme on one-host ports is minimal. The advantages of this scheme easily out-weight the extra overall state maintained, and the schemes relative complexity.

The statements made in this section will be confirmed in Chapter 5, using simulation and measurement results.

Chapter 5

Evaluation

If the facts don't fit the theory, change the facts.

- Albert Einstein, 1879-1955

To give an indication of the effectiveness of the approaches described in Chapter 4 for managing anomalous ARP behavior, and confirm claims made in that section regarding the relative strengths and weaknesses of the various approaches, we carried out a series of simulations.

A network switch was modeled and extended with the probabilistic dropping strategies. Three million ARP requests, that were taken from the traces acquired from networks *N2* and *N3* during our network analysis, were passed to the modified switch. We also ran simulations that included a further twenty five hosts generating synthesised network scans for network *N3*. Information regarding the traces used is presented in Table 5.1. Note that the average request rate for network *N2* is considerably higher than *N3* – this is because the trace for *N2* was taken two weeks after the breakout of an Internet worm, with some hosts still being infected. The simulations were run multiple times for each of the ARP request traffic management schemes, with different random number generator seeds, and for different α and threshold values. Entries in the table of time-stamps were set to timeout after 30 seconds. Results presented are averages of those runs. Various network topologies were simulated, including single-switch topologies and star-like topologies for each of the switch types.

Network	Average (req/sec)	Maximum (req/sec)	Number of hosts
N2	1639	2958	887
N3	95	2207	3061
N3 + 25	142	2393	3086

Table 5.1: Network information for traces used in simulations

5.1 ARP Throttling

To determine if the ARP management strategy correctly carries out its most fundamental role, i.e. that of limiting ARP request traffic volume to a certain predefined value representing a switch's available ARP request forwarding resources, we looked at the number of requests received and requests forwarded per second. Figures 5.1, 5.2, 5.3, and 5.4, plot these for single-switch topologies for each of the ARP management schemes and a threshold value of 128 requests/sec.

The graphs clearly show that the maximum number of forwarded ARP requests is severely limited, therefore limiting the resource consumption of ARP requests sent by malicious and misconfigured software, both on a network's end-systems and internal devices. As a consequence, the network remains operational under those extreme circumstances, e.g. when new Internet worms are propagated. In Figure 5.1 (scheme *SP*) there are several spikes in the number of received requests. During the periods of increased ARP traffic, our switch forwards approximately the same number of requests as before. The same can be seen in Figure 5.2 (scheme *SH*). This confirms that these schemes are successful in limiting the resource consumption of ARP broadcasts. For the schemes with port scopes, however, we see a slight increase in the number of forwarded messages for sustained periods of high request rates. This is because the port where the schemes are running (i.e. threshold has been reached) are using their entire amount of resources allocated for ARP forwarding, increasing the overall forward rate somewhat.

In the graph for the scheme with switch scope and per-host stream identifiers (Figure 5.2), the same decrease in forwarded messages can be seen. However, for more transient spikes, the number of forwarded messages closely follows that of received messages. This is due to transient spikes being caused by a large number of hosts broadcasting ARP requests towards a single destination at the same time. This behavior was described in Chapter 3. Each of the broadcasting hosts has a very low instantaneous request rate, and therefore a high probability of having its requests forwarded. For example, at roughly the 34000th second, 1471 requests are received, 1353 of which were sent towards the same destination. Only 120 of the 1471 requests had high instantaneous rates, causing 63 of them to be dropped (about 53%). For the graph for scheme *PH* (Figure 5.3), the same transient spikes occur. When comparing this with the schemes *PN* and *SP*, we see that those schemes do not forward the legitimate requests during transient spikes. This supports our claim that per-host stream identifiers are more suitable to distinguish between legitimate and anomalous ARP requests.

5.2 Rate of dropped requests

To determine if the ARP management strategy correctly drops ARP requests from hosts that have the highest request rates, we looked at the percentage of requests dropped for a host against its maximum instantaneous request rate. Recall that the instantaneous request rate for a host is used to determine the probability that an ARP request will be dropped (see Equation 4.1). Figures 5.5, 5.6, 5.7, and 5.8 plot these values for all the

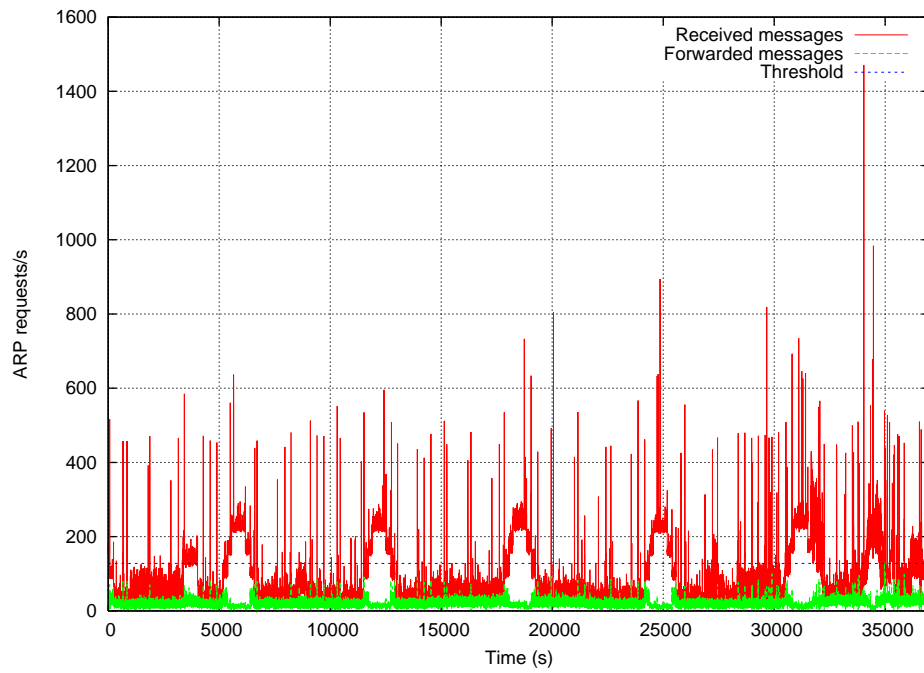


Figure 5.1: Number of received requests and forwarded requests per second on network $N2$ using approach SP

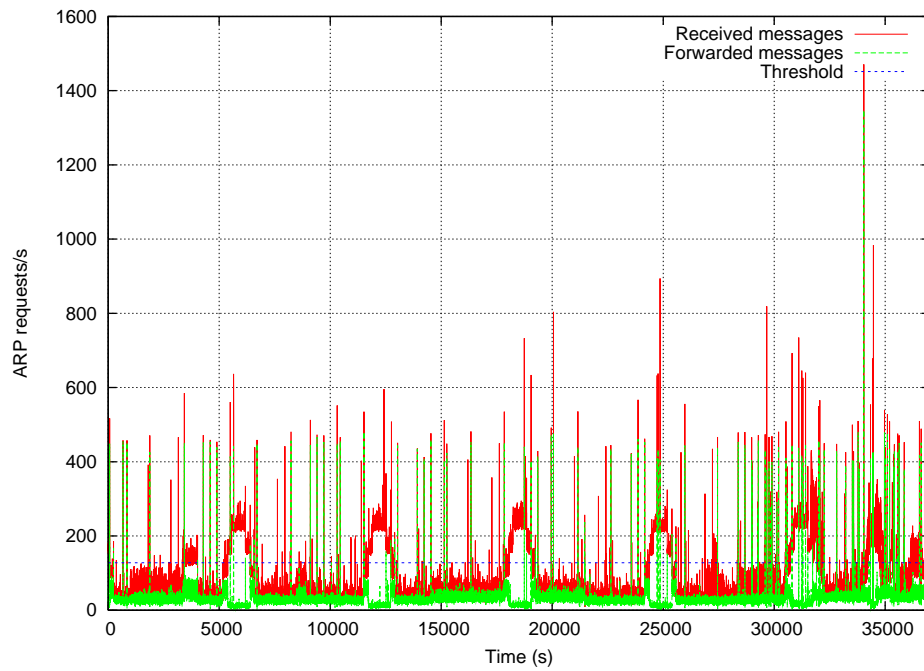


Figure 5.2: Number of received requests and forwarded requests per second on network $N2$ using approach SH

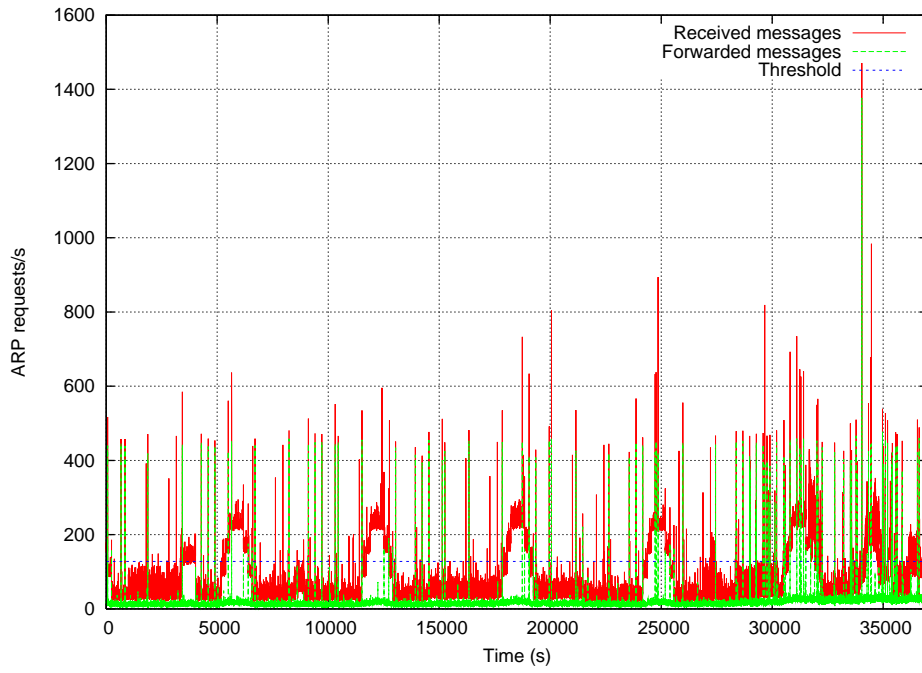


Figure 5.3: Number of received requests and forwarded requests per second on network $N2$ using approach PH

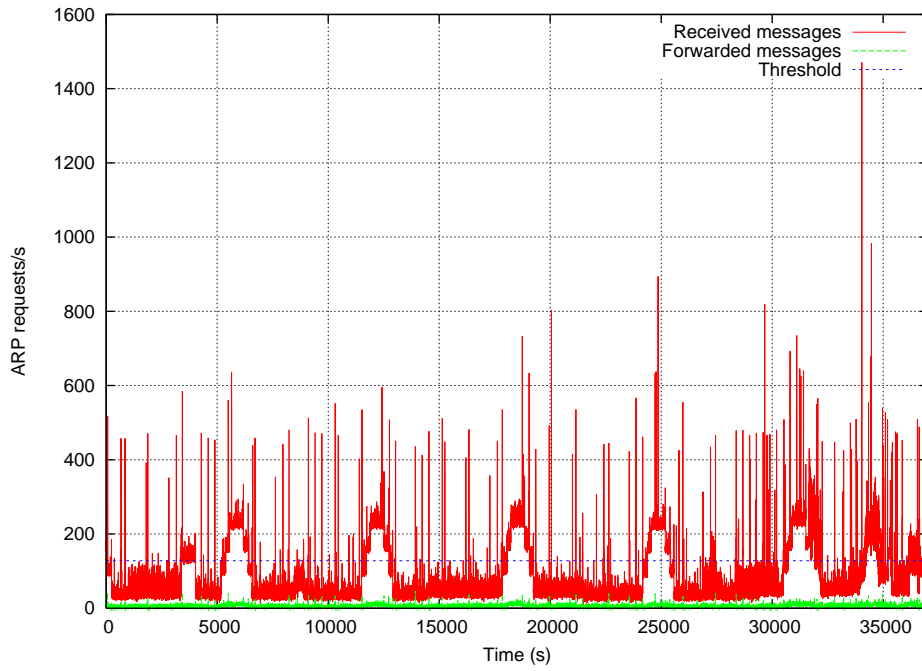


Figure 5.4: Number of received requests and forwarded requests per second on network $N2$ using approach PN

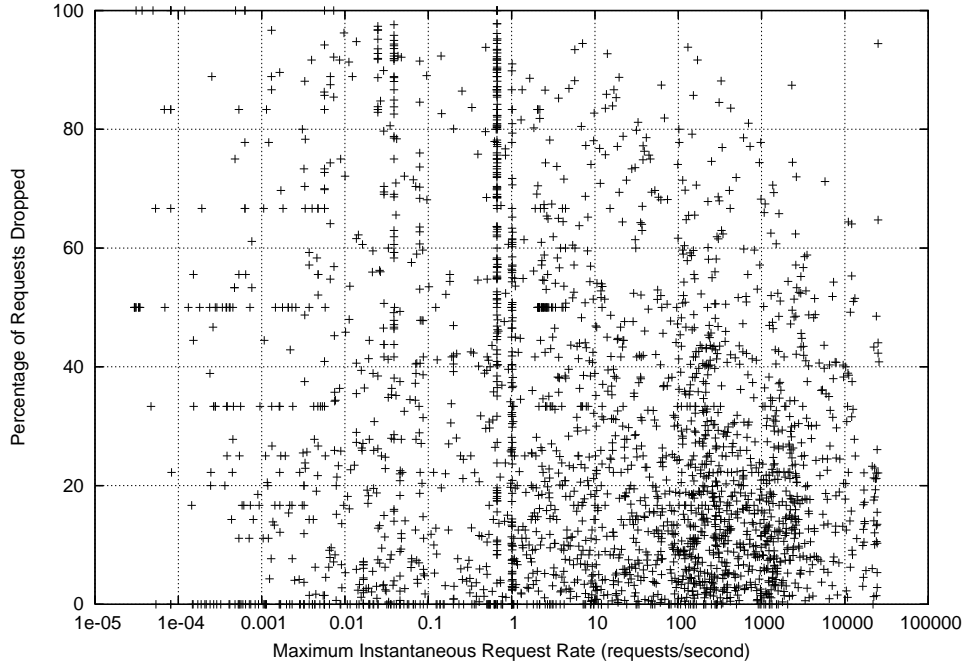


Figure 5.5: For switch *SP* on network *N2*, the percentage of requests dropped against the maximum instantaneous request rate for each host on network

schemes with traces taken from network *N2*. The threshold value t was set to 512 requests/second.

These figures suggest there is a significant difference in the schemes' performance. The schemes that use host stream identifiers drop a relatively low percentage of low-rate ARP requests, while the schemes that use port stream identifiers in the switch scope scheme, or do not identify streams in the port scope scheme, do not distinguish between low-rate and high-rate requests, i.e the probability of these requests being dropped is equal.

Figures 5.6 and 5.7 show that for high request rates, some hosts have a relatively low percentage of their requests dropped. There are a number of causes of this. For example, over an prolonged period a host may generate requests when the management scheme is not invoked (i.e. during periods when the aggregate request rate at the switch is below the threshold), therefore causing a relatively small percentage of its total number of requests to be dropped. Additionally, a host may have a high maximum request rate, while transmitting most of its ARP requests at a low rate. The above graphs indicate that schemes *SH* and *PH* are biased towards dropping requests from high rate request streams, while the rate of the generating host is not a factor in the other schemes, resulting in a lot of legitimate requests being dropped. However, this does not mean that these schemes are useless, as the last section showed they were effective in performing their most basic task, i.e. limiting the resources used in ARP forwarding.

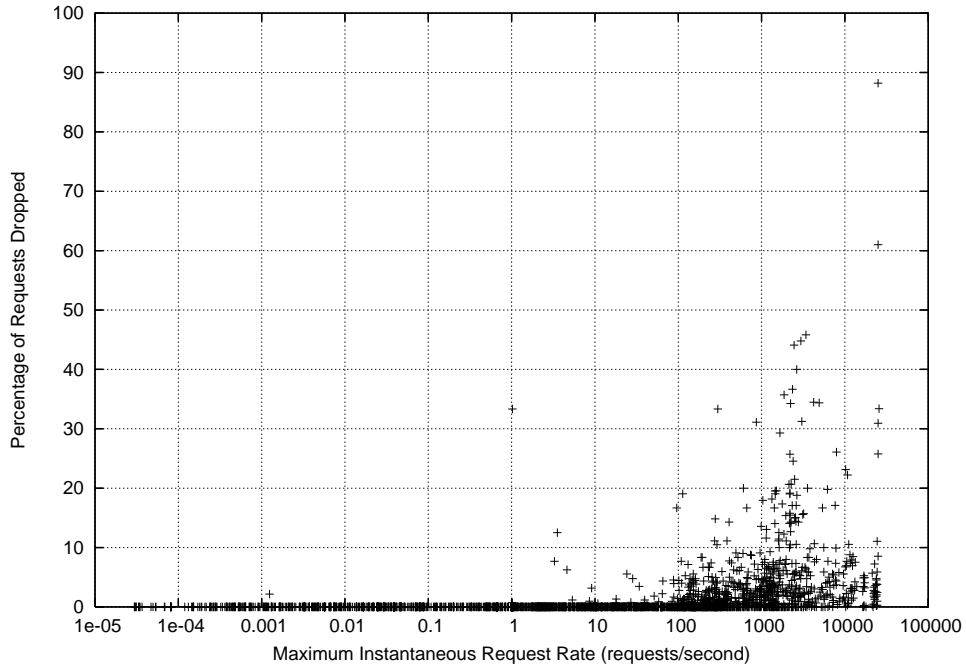


Figure 5.6: For switch *SH* on network *N2*, the percentage of requests dropped against the maximum instantaneous request rate for each host on network

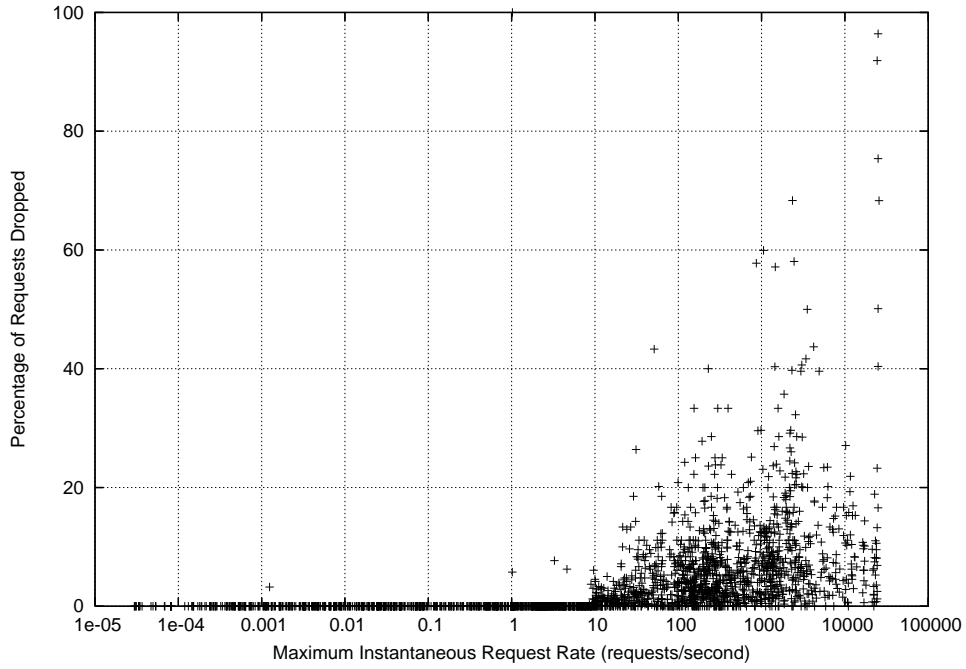


Figure 5.7: For switch *PH* on network *N2*, the percentage of requests dropped against the maximum instantaneous request rate for each host on network

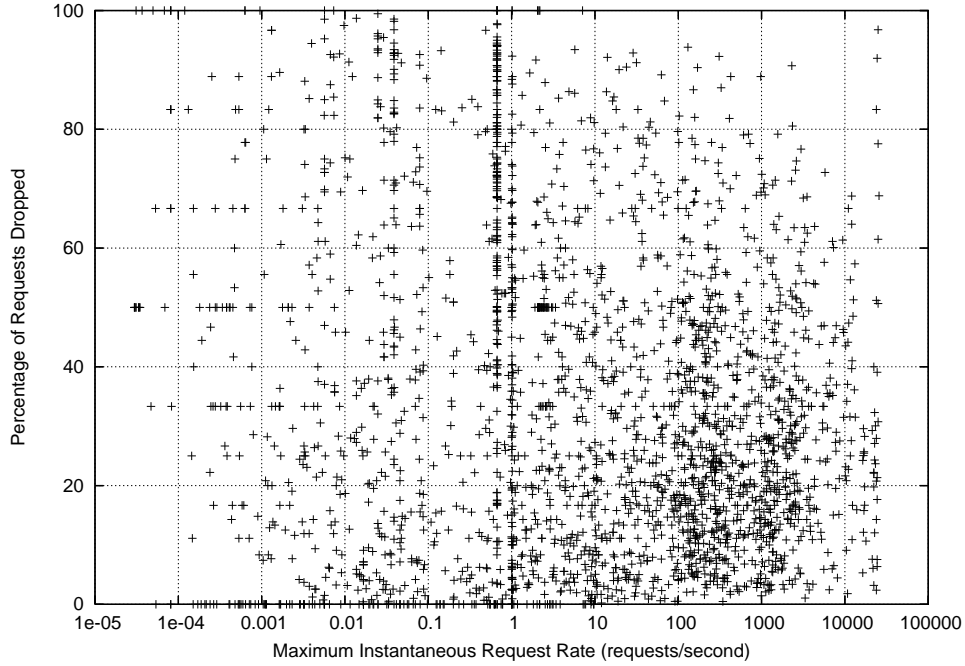


Figure 5.8: For switch PN on network $N2$, the percentage of requests dropped against the maximum instantaneous request rate for each host on network

5.3 Normal vs. anomalous requests

The probabilistic dropping scheme could discard *normal* ARP traffic. To understand to what extent this could occur, we conducted simulations that categorised requests as either normal or anomalous, and looked at the percentage of requests dropped for each category. There are two behaviours we used to determine whether an ARP request was anomalous:

1. A request was part of a sequential network scan. In other words, a request was part of a series with the same source address and a sequentially increasing target address.
2. A request was targeted at an unbound network address. This form of request was found to be generated by network $N3$'s gateway router caused by external network scans, and incorrectly configured devices.

Note that while the likelihood is very high that ARP requests tagged as anomalous by this categorization are indeed anomalous, many anomalous requests could still be categorized as normal (for example, abnormally high request rates towards bound addresses).

Figures 5.9, 5.10, and 5.11 show the percentage of normal and anomalous ARP traffic that was dropped over the length of the simulation with different threshold values for the three network traces used. For network $N2$ (shown in Figure 5.9), it can be seen that the dropping scheme discards a significant percentage (almost 100%) of the requests classified as anomalous and approximately 30% of the network traffic that is classified as normal, for threshold values of 128 through to 1024 requests/second. A 30% drop rate of normal traffic appears unacceptably high.

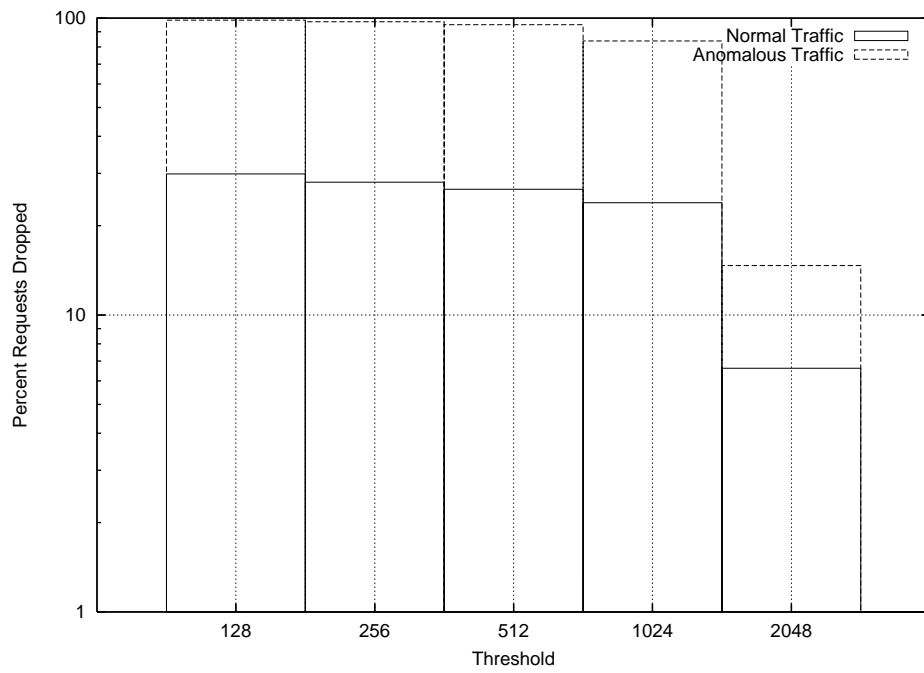


Figure 5.9: Percentage of the normal and anomalous ARP requests dropped for network $N2$ with different threshold values.

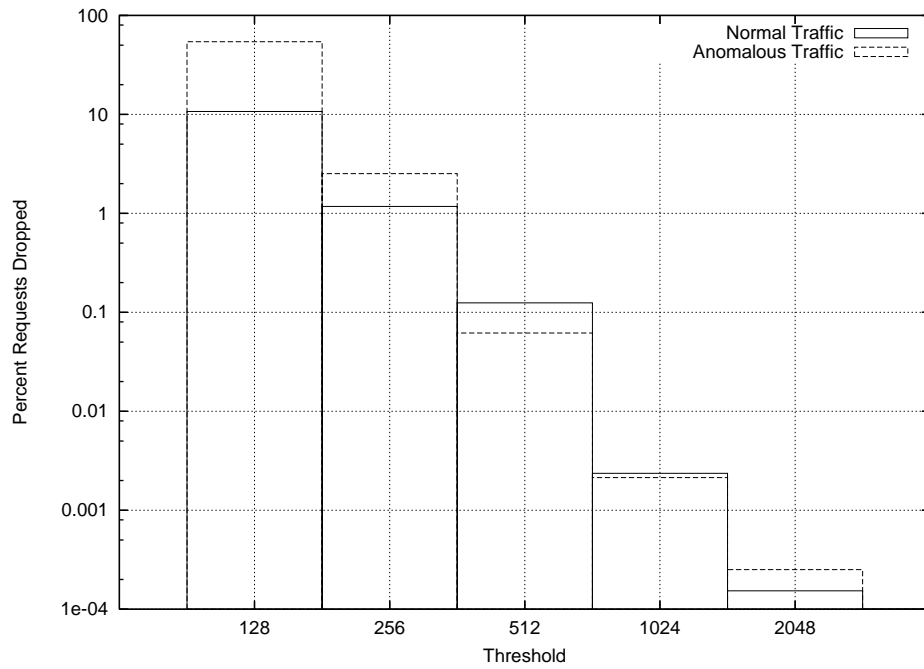


Figure 5.10: Percentage of the normal and anomalous ARP requests dropped for network $N3$ with different threshold values.

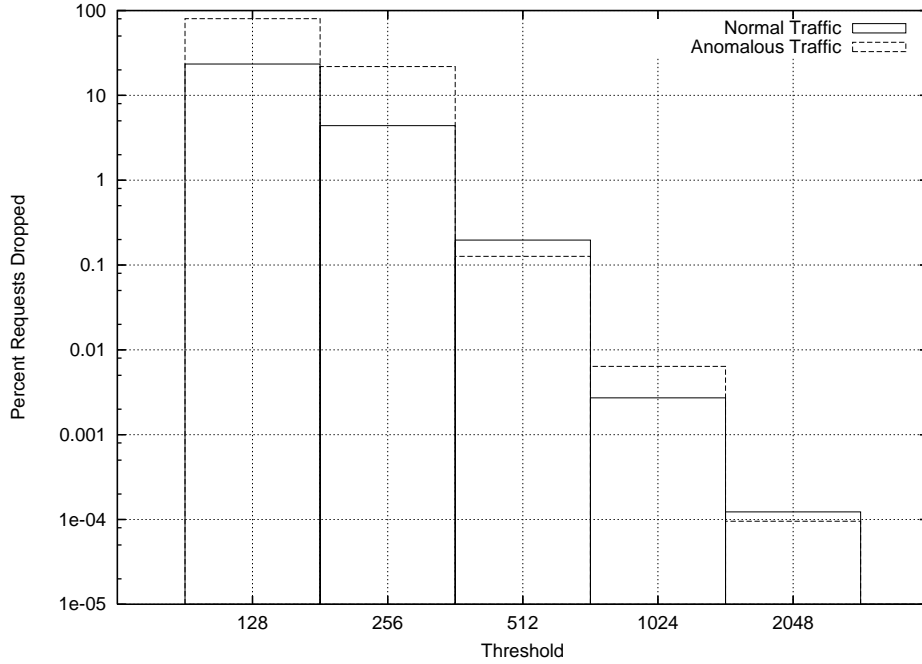


Figure 5.11: Percentage of the normal and anomalous ARP requests dropped for network N_3 with synthesised scans with different threshold values.

To understand this result further, we looked at the percentage of dropped normal ARP requests that were sent by hosts whose ARP request stream only contains requests categorised as normal – we found that none of these requests were dropped. Sources of “purely” normal ARP request streams are very likely to be hosts that only engage in normal communication patterns resulting in low ARP request rates. The zero drop observation for such hosts is very significant because it shows that our scheme is very good at discriminating positively for well behaved hosts. We can therefore be confident that the normal traffic dropped by our scheme is either anomalous requests that have been misclassified by our simple categorisation, or normal requests from hosts that also generate anomalous ones.

This is another piece of evidence that our proposed scheme is biased against ARP request streams from hosts that contain anomalous traffic. This is, in most cases, a very good property because their normal communications would be hindered, providing both an incentive to quickly patch a host infected by malicious scanning software and an indicator of the infection in the first place.

Unfortunately, this could be problematic for the ARP request stream sent by the gateway router, as this request stream can contain a significant amount of anomalous requests (see Chapter 3.2). However, in Chapter 2, we discuss simple methods to reduce the ARP request rate caused by external scans through the gateway. The use of such methods, along with appropriate configuration of the gateway (e.g., ARP table size large enough to contain most, if not all, address bindings for the hosts in the LAN, ARP table entry expiration timers large enough and/or the use of ARP table entry verification on expiration) as well as opportunistic ARP cache population on receiving requests, can ensure that the ARP re-

quest rate generated by the gateway is “normalised”, so that both inbound and outbound communications can occur unhindered by our proposed scheme.

Another observation from Figures 5.10 and 5.11, is that for some thresholds the scheme drops a higher percentage of normal requests than anomalous – this appears to be worse than just dropping a random proportion of all the requests. For these thresholds, the scheme is dropping a relatively small number of requests that are from hosts that have a high request rate. For example, for a threshold of 512 for network *N3* (shown in Figure 5.10), the scheme is dropping approximately 0.1% (811 requests) of the total number of normal requests and the average instantaneous request rate for hosts that generate these normal requests is 933 requests/second. Again, this suggests that our definition of normal ARP requests is not strong enough and includes ARP requests generated at an abnormally high rate.

The effect of different threshold values can be seen in Figures 5.9, 5.10, and 5.11. For network *N3*, whose average request rate is 95 requests/second, the percentage of dropped requests falls as the threshold becomes greater than the average request rate. This is to be expected, as the scheme will be invoked less often and dropping will be less aggressive when it is executing. For network *N2*, when the threshold is well below the average request rate (1639 requests/second), the percentage of requests dropped remains approximately constant. This is due to the threshold being so low that each host has a very small fair share of the switch’s resources, so that those generating requests at high rate are aggressively controlled by the scheme.

Figure 5.12 shows the cumulative distribution function (CDF) of the number of dropped requests as a function of the instantaneous request rate of their ARP request streams when they were dropped. Keeping in mind that a sustained ARP request rate of 1 request/second per individual host can already be considered as a high rate, we see that over 90% of all request drops occur at high and very high rates, while the dropping of requests that are part of ARP request streams with a low (normal) rate is well below 1 request/second and is an extremely rare event.

5.4 Retransmissions

As mentioned in Chapter 2.1, if a host does not receive a response to an ARP request it has broadcast, it will retransmit the request after a timeout period. Retransmissions may be sent a number of times if there continues to be no response. We implemented this behaviour for our simulations. More specifically, the behaviour we implemented, which was observed on a Linux-based host, was to retransmit up to three requests at one second intervals. Our aim was to determine to what extent a host will have a request dropped and any subsequent retransmissions. If all requests are dropped, this will render the host unable to send frames to the targeted host.

Figures 5.13, 5.14, and 5.15 show both the percentage of normal ARP requests and the subsequent retransmissions that were dropped. We observed that if a normal packet is dropped, there is a high probability that subsequent retransmissions will suffer the same

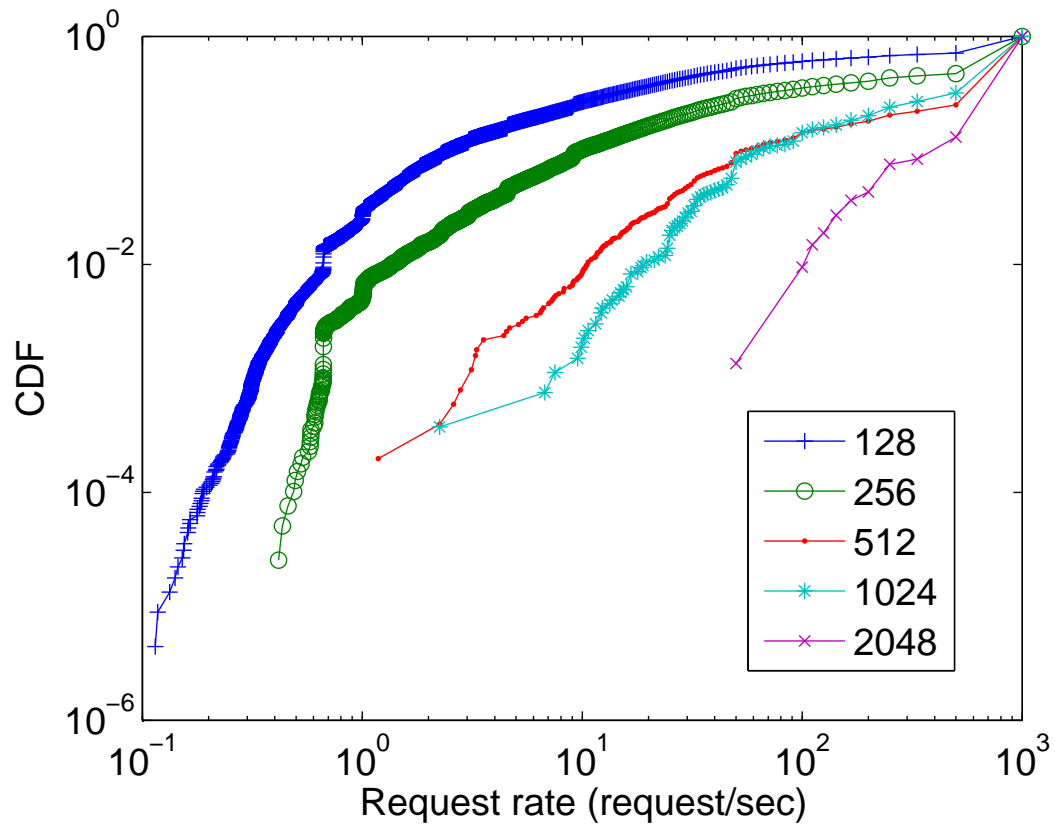


Figure 5.12: CDF of the number of drop requests across all three networks for various threshold values.

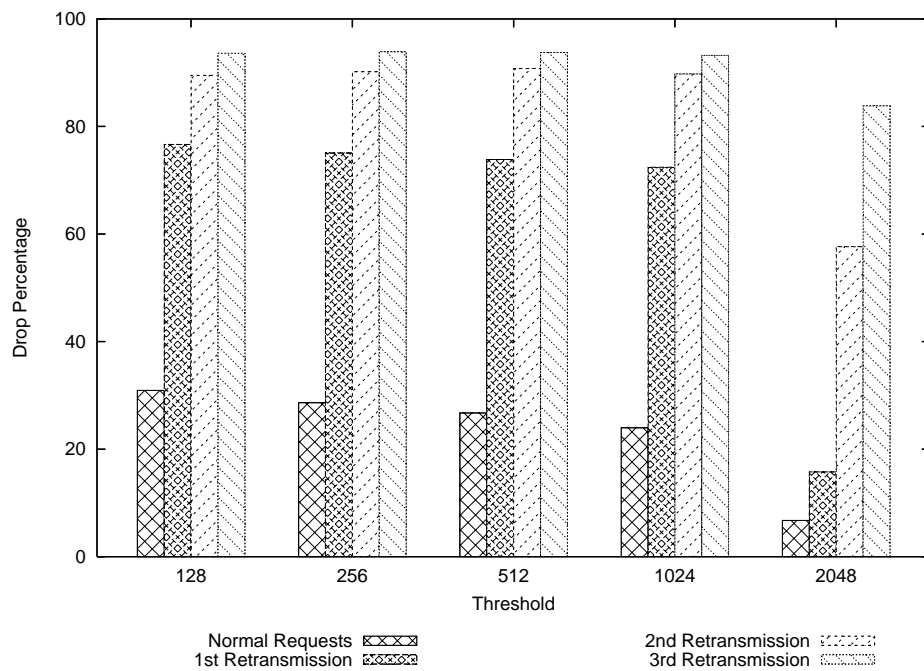


Figure 5.13: Percentage of the normal requests and retransmissions that are dropped for network N_2

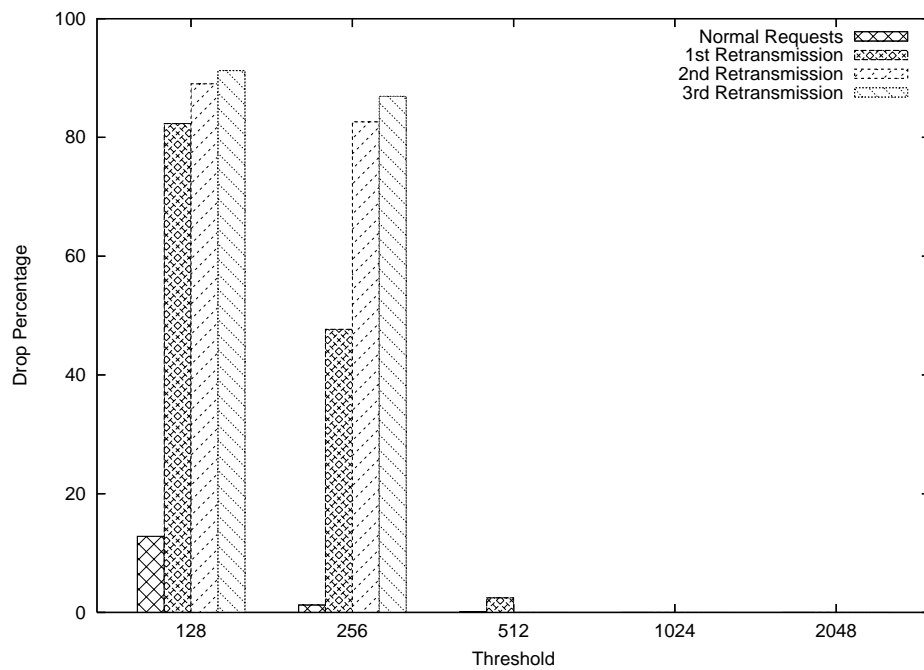


Figure 5.14: Percentage of the normal requests and retransmissions that are dropped for network N_3

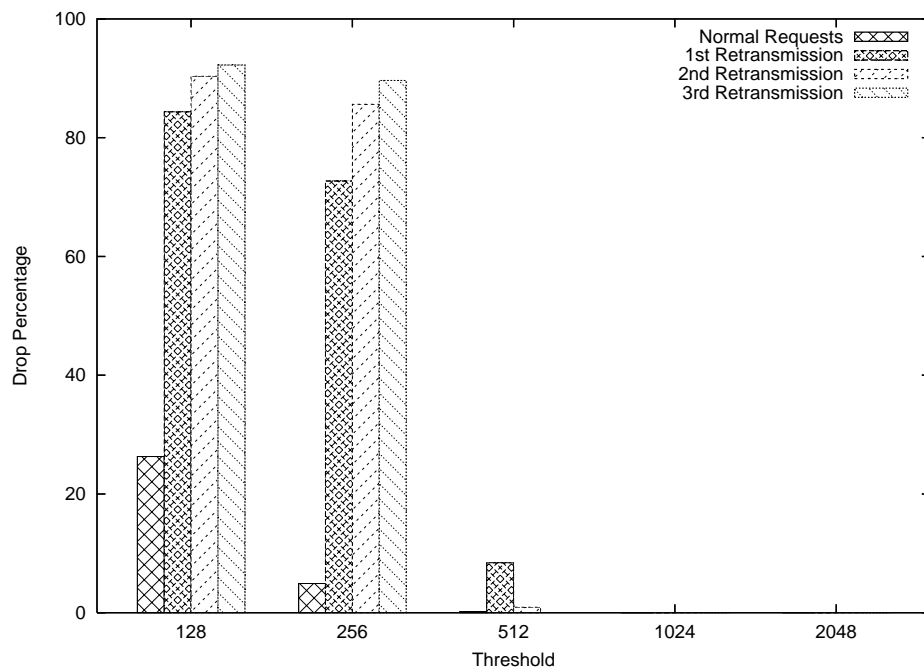


Figure 5.15: Percentage of the normal requests and retransmissions that are dropped for network N_3 with synthesised scans

fate. Again, we believe this is largely due to the fact that normal requests are being transmitted by hosts that are sending at abnormally high rates. If this were not the case, the instantaneous request rate for these hosts would be approximately 1 request/second – considerably lower than the 933 requests/second average mentioned earlier – and would have a low drop probability.

Our evaluation demonstrates that the ARP management scheme is effective at dropping anomalous requests that exhibit a sustained high rate. Although our results show that in some cases a significant percentage of normal traffic is being dropped, our analysis shows that these are either part of an overall anomalous request stream or that these requests have been misclassified by our rather simple categorisation. We conclude that non-malicious and correctly configured hosts will have an extremely low probability of their ARP requests being dropped by our scheme, while anomalous requests are effectively controlled.

Chapter 6

Implementation on the Intel IXP1200 Network Processor

When first programming on the IXP12xx, accomplishing seemingly trivial tasks such as receiving or transmitting packets has been known to invoke an all-out engineer's victory dance.

- Johnson and Kunze, "IXP1200 Programming"

As part of this project, we have designed and implemented the ARP management scheme on the Intel IXP1200 network processor. This serves as a proof of concept of our proposed scheme, as well as to give the student a valuable experience in designing and programming embedded applications in an environment where concurrency, robustness, and performance are of extreme importance. Finally, to understand the cost of running the scheme, what effects the scheme has on non-ARP traffic, both under light ARP load and high ARP load, etc. This process included designing and implementing a standard layer two switch, and then extending the switch with our ARP management facilities.

6.1 Intel IXP1200

Network processors are a special piece of hardware that enable network service providers to add, through software, cutting edge services while maintaining a high throughput and low latency. Further, network processors allow providers to deploy new services more quickly than if the service was to be implemented in hardware. Intel's Internet eXchange network Processor (IXP) is a family of such network processors.

The IXP12xx contains a variety of hardware units and co-processors specifically tuned to network programming. Further, the IXP12xx contains a general-purpose core processor. In the case of the IXP1200, there are six co-processors, so called micro engines, in addition to a StrongARM core processor. Each of the micro engines supports four hardware threads, i.e. can run up to four threads with zero cost context switches. This is accomplished by using four sets of registers. Additionally, the IXP1200 contains units to

perform common tasks in network service development, such as generating hash values. Finally, the IXP1200 contains three memory banks, each varying in size and speed.

This specialization allows network service providers to separate responsibilities into self-contained components, each designed and run on a specific part of the platform. For example, the most commonly executed fast data path through the network processor might be implemented using the very fast micro engines, while exceptional slow data path processing is performed using the general-purpose core processor. The fast data path could be further split among the micro engines, thus controlling the resources spent on each component precisely. This all enables the service provider to develop new services more efficiently and gain higher performance than might be expected with a general-purpose processor. Coincidentally, this is the approach we have chosen in our design of an ARP managing switch for the IXP1200 platform (see Appendix A).

6.2 Evaluation

As stated earlier, we have developed an IXP1200 version of our ARP management scheme. The design is similar to what is described in Chapter 6.1, i.e. a standard link-layer switch is implemented on five micro engines, while the slow data path ARP management scheme is run in the StrongARM core processor. The design is described in more detail in Appendix A. In this section we evaluate the ARP-managing, IXP-based network switch, both analytically and through measurements.

First, let us adopt an analytical approach. When inspecting the design and code we see that the only place where normal traffic is affected is in the start of the data path where the path branches to the bridging unit as part of the fast path on one hand, and the ARP managing scheme on the other. This is achieved using a single conditional that checks whether the frame in question is an ARP request frame. From then on, normal traffic is unaffected by the presence of the ARP managing scheme. Further, since the ARP managing scheme is run entirely on the core processor, it is not using up processing resources that would otherwise be used for fast data path processing. As the core processor uses the same memory unit as the micro engines, the extra memory copy of the ARP request frame handle required uses a limited amount of common resources.

All ARP requests, however, suffer a penalty hit by going through the management scheme on the core processor. While the scheme is not running, this consists of a single extra copy of the frame handle (from the management scheme to the bridge queue) and updating the scheme-wide average request rate. Considering that ARP requests are broadcast, and therefore copied to each transmit queue, a single extra copy is relatively cheap, and more so as the number of ports grows. The calculations required for rate calculation are relatively expensive, as the core RISC processor does not contain a floating-point arithmetic unit. We have developed an approximation of the calculation presented in Chapter 4, that uses integer multiplication and bit-shifting, making the calculation cheap enough so that only the memory copy matters.

When the scheme is running, however, the cost is increased by the cost of a hash-map lookup, rate update for the stream identifier, and a random number generation. This cost is small compared to what we gain compared to not managing ARP traffic.

We have measured all the previously mentioned parts of the data path. The experimental setup consisted of a machine running a packet generator connected to one of the IXP's ports, a second machine running a traffic collector on another port, and traffic sinks connected to the remaining six ports. Packets were timestamped as they entered the switch and again when they had been copied to hardware queues on the outgoing ports. Steps have been taken to correct the measurement results to remove the effects of timestamping forwarded packets. Packets of various sizes were sent through the switch, enabling us to estimate the relative costs compared to packet sizes.

To obtain a baseline necessary for comparison and estimating the relative cost of the ARP management scheme, we measured fast data path forwarding using our switch implementation. The results for various packet sizes are presented in Figure 6.1. The graph shows that forwarding cost increases in steps with increasing packet size, where a linear growth is perhaps expected. This is because the IXP architecture copies 64 byte of packet data at a time from and to port hardware queues, and more such copy operations are required for larger packets. The most important measurement for comparison purposes is the cost of forwarding a minimum sized packet on the fast path, or about 1700 clock cycles on average.

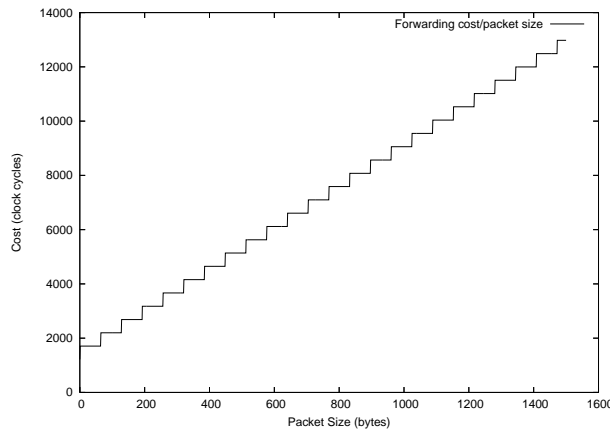


Figure 6.1: The cost of forwarding packets using the fast path increases as a function of their size.

To obtain an understanding of the cost data path branching incurs on normal traffic, we timestamped packets before and after the conditional branching statement. The results show that branching a single message costs about 70 clock cycles on average. When this is compared to the costs of forwarding data packets on the fast path, the cost ranges from 4.11% for minimum sized packets, to 0.53% for maximum sized packets.

To measure the extra cost of running the ARP management scheme under normal conditions, i.e. no malicious or misconfigured software is exposing abnormal behavior, we measured ARP packets, at exactly the same places in the code as the fast path forwarding measurements, i.e. before the branch point and after the ARP requests had joined the fast path. Our measurement show that the extra cost incurred on the ARP requests is about

700 clock cycles, or about 41% of fast data path forwarding of minimum size packets. It should be noted though that ARP requests would receive broadcast treatment on the fast data path, so that the extra cost of sending ARP requests through the management scheme is considerably less than 41%. We have determined that of the roughly 1700 cycles that fast path uses to forward a single minimum size packet, about 740 of those are spent on transmitting the packet on a single queue. Broadcasting on all but the incoming port on a eight port switch therefore costs about 5180. The cost of processing a single minimum sized packet is therefore about 6140 clock cycles. The extra cost associated with the ARP management scheme compared to that cost is about 11%. This percentage decreases as the number of ports increases. On a 32 port switch, the extra cost is less than 3%. It may be argued that the ARP management scheme somewhat reduces the ARP throughput of a switch, but it is our claim that excessive ARP traffic is undesirable and the threshold of the scheme should be configured so that the resources used for ARP forwarding are only a fraction of the switches total ARP forwarding capacity.

Chapter 7

Conclusions

I feel that if a person has problems communicating the very least he can do is to shut up.

- Tom Lehrer

In this thesis, we set out to investigate the claim that broadcast based protocols imposed scalability limitations to Ethernet networks. We studied broadcast traffic on three sizeable Ethernet networks, in particular traffic generated by the Address Resolution Protocol. In summary, we found that under normal conditions ARP did not generate an excessive amount of network load and is not a scalability concern.

However, the presence of malicious software and external network scans can cause serious performance problems, and easily render a network non-usable. We have presented a family of simple, yet very effective schemes to control the amount of ARP traffic present in the network. Our schemes efficiently isolate anomalous ARP traffic, while leaving normal ARP traffic unaffected, which is a very desirable property. It should also be noted that thanks to the opportunistic ARP mapping and cache population advised by the ARP standard, a node generating normal ARP traffic will be able to instantiate communication unhindered with a host or gateway router generating anomalous ARP request patterns.

As stated earlier, our schemes are complimentary to other approaches that try to limit the effects of scanning worms located outside the local-area network, such as black-hole routers and firewalls. However, so far as we know, our solution is the only proposed defense against performance threats of malicious software, once it is inside the local-area network.

Our family of ARP management schemes contains multiple variants, each requiring different amount of memory and processing resources from the hosting network switch, as well as varying in its performance in filtering out anomalous ARP traffic while leaving normal traffic unaffected. Each member of the family of schemes is applicable in different scenarios, and can be combined in a single network topology to get an optimal balance of performance vs. cost.

From an implementation point of view, it is important to note that our scheme operates on the ARP request traffic only. In switches that adopt a *store-and-forward* architecture, the

frame-type of Ethernet frames can be used to easily “filter out” ARP requests. For those that implement a *cut-through* forwarding strategy, all broadcast traffic can be filtered off the fast-path for further processing. As broadcast traffic on a LAN is typically used for control purposes, our scheme should have no impact on the forwarding capabilities of switches for data traffic.

Furthermore, we believe the additional state required at a switch to implement the strategy proposed here is not prohibitively large. On a per host basis, only the time-stamp of the previous ARP request received from a host is necessary. A small addition to a switch’s forwarding table. For the entire switch, we need only to maintain state regarding the threshold value (a weighted moving average) for the whole switch and a counter to help compute a fair share. With a ball-park figure of approximately \$10 per megabyte of memory, state for several millions of hosts can be accommodated cheaply, as a time-stamp can be implemented using very few bytes.

A positive side-effect of our proposed scheme is the role it plays in slowing the propagation of viruses by capping their probing rate. Also, the approaches that use per-source stream identifiers discourage the use of MAC address masquerading while probing, as it maintains a record of the number of sources seen to compute a fair ARP rate.

Although this thesis addresses a real potential issue in very large-scale wired Ethernet networks, it can find applications in improving performance of other types of Ethernet-based networks, e.g. to guarantee a fair forwarding resource consumption in wireless multi-hop networks.

We believe that our results on the analysis of ARP traffic can be projected onto other broadcast based control channel protocols, i.e. that such protocols can in general be designed such that they do not impose scalability limitations on Ethernet networks. Further, that similar randomized local network-resident approaches can be used to alleviate similar performance issues caused by these protocols.

Future research on the ARP management schemes includes studying the composition of multiple scheme types in a single topology, the effect of the α -value on the algorithms performance, and the application of similar approaches on other similar problems in the Ethernet domain and in similar networks, e.g. multi-hop WiFi networks. Finally, further work is need to formally establish that ARP is representative of control channel broadcast protocols in general, and to generalize the schemes to manage all such protocols.

Bibliography

- The 100x100 clean slate project.* (2003, March). <http://www.100x100network.org/>. 100x100 Research Group.
- Ármannsson, D., Smith, P., Hjálmtýsson, G., & Mathy, L. (2005, October). Controlling the effects of anomalous ARP behaviour on ethernet networks. In *Conext 2005*. Toulouse, France.
- BellSouth Metro Ethernet.* (2006, December). <http://www.bellsouthlargebusiness.com>. BellSouth.
- Carl-Mitchell, S., & Quarterman, J. S. (1987, October). *RFC 1027: Using ARP to implement transparent subnet gateways.*
- Chen, Z., Gao, L., & Kwiat, K. (2003). Modeling the Spread of Active Worms. In *IEEE Infocom 2003*.
- Droms, R. (1997, March). *RFC 2131: Dynamic Host Configuration Protocol.*
- García, R., Duato, J., & Silla, F. (2003). LSOM: A link state protocol over mac addresses for metropolitan backbones using optical ethernet switches. In *Nca* (p. 315-321).
- Greene, B., & McPherson, D. (n.d.). *Sink holes: A Swiss army knife isp security tool.* http://www.arbornetworks.com/downloads/research36/Sinkhole_Tutorial_June03.pdf.
- Heberleid, L. T., Dias, G., Levitt K, B. M., Wood, J., & Wolber, D. (1990). A Network Security Monitor. In *Proceedings of the IEEE Symposium on Research in Privacy*.
- Hjálmtýsson, G., & Ármannsson, D. (2005, June). *CLIP: A CLI Parsing Tool for Parsing and Analyzing Router Configuration Files.*
- IEEE. (2002). *IEEE 802.1s Multiple Spanning Trees.* <http://www.ieee802.org>.
- Microsoft Windows Update.* (n.d.). <http://windowsupdate.microsoft.com>.
- Myers, A., Ng, E., & Zhang, H. (2004, November). Rethinking the Service: Scaling Ethernet to a Million Nodes. In *ACM SIGCOMM HotNets 2004*. San Diego, CA, USA.
- Perlman, R. J. (2004). Rbridges: Transparent Routing. In *Infocom*.

- Plummer, D. C. (1982, November). *RFC 826: Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware*. <http://www.ietf.org>.
- Roesch, M. (1999, November). Snort: Lightweight Intrusion Detection for Networks. In *Proceedings of the 13th conference on system administration*.
- Sharma, S., Gopalan, K., Nanda, S., & Chiueh, T. (2004, March). Viking: A Multi-Spanningtree Ethernet Architecture for Metropolitan Area and Cluster Networks. In *Ieee infocom 2004*. Hong Kong, China.
- Symantec. (n.d.). *W32.Blaster Worm*. <http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.worm.html>.
- Twycross, J., & Williamson, M. (2003). *Implementing and testing a virus throttle*.
- Weaver, N., Paxson, V., Staniford, S., & Cunningham, R. (2003, October). A Taxonomy of Computer Worms. In *In the first ACM Workshop on Rapid Malcode (WORM)*. Washington DC, USA.
- Williamson, M. (2002). *Throttling viruses: Restricting propagation to defeat malicious mobile code*.
- Yipes. (2006, September). <http://www.yipes.com>. Yipes Enterprise Services, Inc.
- Zec, M., Mikuc, M., & Zagar, M. (2002). Estimating the Impact of Interrupts Coalescing Delays on Steady State TCP Throughput. In *Proceedings of the 10th softcom 2002*.

Appendix A

Design of a ARP Managing Switch for the IXP Network Processor

The ARP managing link-layer switch has been implemented using the Intel IXP1200 network processor platform. As described in Chapter 6.1, the platform contains six micro engines, each capable of running four hardware threads, and a general-purpose core RISC processor. Our design runs the fast data path forwarding, i.e. the normal operations of a link-layer switch, on the micro engines, while running the ARP management code on the core processor. An overview of the design is given in Figure A.1.

Communication between any two of the micro engines, and between a micro engine and the core processor, go through shared memory. In addition, signals are used for synchronization. Each of the micro engine threads can send and receive signals. The core can signal the micro engine threads, but is unable to receive such signals.

Two micro engines run code that receive frames from the hardware ports. The use of the eight hardware threads of the two micro engines allow us hide latencies associated with IO operations, such as memory access, i.e. while one thread waits for the memory unit to signal the completion of an IO operation, another thread can run, maximizing processing power utilization. The receive-engines copy incoming frames from the port hardware queue into SDRAM memory, which is the largest, and slowest, memory bank on the platform. A handle of the frame is then stored in one of two queues in SRAM memory. ARP request handles are stored in the ARP queue, serviced by the ARP management code running on the StrongARM, while other frame handles are stored in the bridge queue.

A bridge micro engine services the bridge queue. It dequeues frame handles from the queue, performs address learning and looks up the outgoing port for the destination MAC address specified in the frame in a hash-table stored in SRAM memory. Address learning consists of adding new entries to that table as new source MAC addresses are discovered. Following the lookup, the bridge engine enqueues the frame handle in one of eight transmit queues corresponding to the switches ports. The transmit queues also reside in SRAM memory. If the destination MAC address is not contained in the hash-table, the frame handle is enqueued in each of the transmit queues, i.e. the frame is broadcast on the

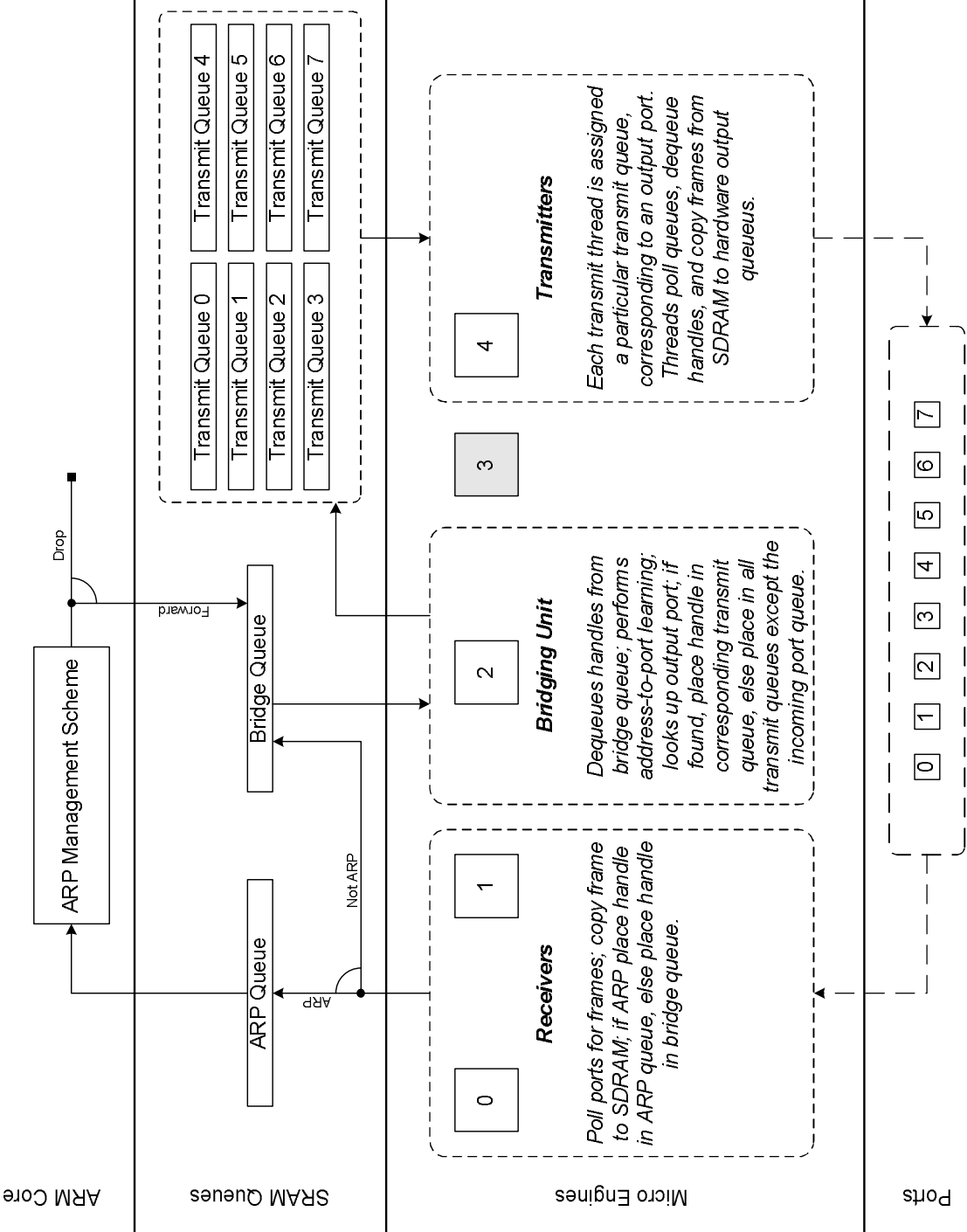


Figure A.1: Overview of the switch design.

Ethernet network. The same is done for frames containing the Ethernet broadcast address in the destination address field.

Of the bridge engines four hardware threads, one acts as a supervisor, i.e. it dequeues handles from the bridge queue and delegates the work of performing the actual bridging to one of the four worker threads. This minimizes the required synchronization when dequeuing from the bridge queue, as there is no contention between threads.

Two micro engines service the transmit engines. Each of the eight hardware threads is responsible for one of the transmit queues. The threads dequeue frame handles from the queues and copy the frame referenced by the each handle to port corresponding to the queue.

The ARP management scheme is implemented as a user-space process on a Linux operating system running on the StrongARM processor. The process creates an instance of a ARP management scheme class using a ARP scheme factory. The type of scheme is determined by a command line option. The process initialized the scheme using parameters from the command line, such as the scheme wide threshold and the alpha value used in average rate calculation. After initialization, normal operation starts. The process polls the ARP queue, dequeues the ARP request frame handles enqueued by the receive-engines, updates the scheme's average rate, determines if the rate breaches the predefined threshold, and if so, makes the request subject to probabilistic dropping. If the random coin-flip determines that the request should be forwarded, or if the scheme is not engaged because the average rate is sufficiently low, the process enqueues the request's frame handle in the bridge queue. At that point, the slow path merges with the fast data path. If the ARP management scheme determines that the request should not be forwarded, the process frees the memory buffer occupied by the ARP request.



REYKJAVÍK UNIVERSITY

HÁSKÓLINN Í REYKJAVÍK

Department of Computer Science
Reykjavík University
Ofanleiti 2, IS-103 Reykjavík, Iceland
Tel: +354 599 6200
Fax: +354 599 6201
<http://www.ru.is>