Controlling the Effects of Anomalous ARP Behaviour on Ethernet Networks

D. Ármannsson and G. Hjálmtýsson
Network Systems and Services Laboratory
Reykjavík University
dadi01@ru.is, gisli@ru.is

P. D. Smith and L. Mathy
Computing Department
Lancaster University
p.smith@comp.lancs.ac.uk,
laurent@comp.lancs.ac.uk

ABSTRACT

There are a large number of large-scale Ethernet-based local and metropolitan area networks in use. A significant reason for this prolific deployment is the relatively simple manner in which they can be configured and deployed. A critical service on these networks, that epitomises the simple nature of Ethernet, 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 intrinsic broadcast capability of Ethernet to determine these mappings. In this paper, we present an analysis of ARP behaviour on three sizable local area networks and show that due to poorly configured or malicious software (e.g. viruses) on hosts, performance issues could arise because of ARP. We also propose a scheme that can be used to manage the effect of the problems identified in our analysis.

Categories and Subject Descriptors

C.2.3 [**Computer Communication Networks**]: Network Operations – *Network monitoring*; C.2.5 [**Computer Communication Networks**]: Local and Wide-Area Networks – *Ethernet*

General Terms

Algorithms, Measurement, Performance, Reliability

Keywords

Address Resolution Protocol, rate limiting, Ethernet performance

CoNEXT'05, October 24–27, 2005, Toulouse, France.

1. INTRODUCTION

Implementing large-scale local and metropolitan area networks using link-layer switching technologies is becoming an increasingly attractive proposition. The simple manner in which such networks can be formed and the increasing capabilities of current switching devices has led to these networks growing significantly in size and sophistication. It is possible today to construct switched Ethernet networks that serve in the order of thousands of systems with relative ease.

A fundamental service on a local area network is one that resolves network-layer identifiers to an associated identifier at the link-layer. The Address Resolution Protocol (ARP) [6] is used in Ethernet networks to discover the 48-bit address associated with a given 32-bit IPv4 address. In short, the protocol functions by broadcasting resolution request messages to all hosts on the local area network. A target host responds with a message that is addressed to the originator. We will describe the operation of ARP in more detail in Section 1.1.

While the operation of ARP is extremely simple and requires no specialised infrastructure support (i.e., servers), the use of broadcasting to resolve address mapping requests has a number of potential problems. In Section 2, we present the results of an analysis of ARP traffic on three relatively large local area networks. Our aim is to identify the main characteristics of the protocol's behaviour and any problems associated with the broadcast nature of ARP. We find that the normal behaviour of ARP on the networks studied is of little concern, but anomalous ARP behaviour caused by malicious hosts or poorly configured servers could prove problematic.

In Section 3, we propose a scheme to address the problems identified in our analysis of the local area networks that limits the rates of abnormally behaving hosts (those that generate an unusually large amount of ARP request traffic) to their fair share of the resources allocated to ARP request forwarding at a network switch. In summary, the scheme probabilistically drops the ARP requests of hosts that use more than their share of the available network resources allocated to ARP forwarding. An initial evaluation is presented, in Section 4, that demonstrates that the approach we propose has the potential to be effective at controlling the effect of anomalous ARP behaviour.

1.1 The Address Resolution Protocol

When a device on a network needs to deliver an IP packet to another device, it needs to frame the packet and address it to the correct Network Interface Card (NIC) for the target host. The Address Resolution Protocol (ARP) is the primary means by which a network device discovers the hardware address associated with the IPv4 address for another device on the local area network.

ARP is a simple protocol – if a device s needs an address mapping for protocol address p_t, it simply broadcasts an ARP request message asking: *Who 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, host t will send a unicast reply stating, *I have address p_t and my hardware address is mac_t*.

As an optimisation, each device maintains an ARP table where they cache address mappings (of the form *(p_i, mac_i)*) obtained by previous address resolution requests. Before sending an ARP request targeted at host d, a host checks its ARP table for an entry for address p_d. Since hosts usually send bursts of packets to other hosts the cache ensures that a host doesn't repeat ARP requests for every single packet. ARP requests are retransmitted a certain amount of times before the target is assumed not to exist on the network.

According to the protocol specification, a target t of an ARP request sent by s should opportunistically store the address mapping *(p_s, mac_s)* in its ARP table. This is done on the assumption that communication is bidirectional and eliminates the need for targets to immediately broadcast ARP requests for originators of requests targeted at them. Entries in the ARP table expire after a given period of time. The protocol specification recommends verifying expired ARP entries directly using unicast, broadcasting only in case of no response to the unicast request. If no response is received from the broadcast request the entry is removed.

2. NETWORK ANALYSIS

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. For one of our networks, the data included periods of ARP flooding where several network devices infected by an Internet worm [9] 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 1: The number of devices and type of each of the three test networks.

The three networks (summarised in Table 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, several of which were still unpatched several weeks after a severe worm infection. 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 a significant overhead 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 behaviour. With an understanding of normal ARP behaviour, we will point out and quantify the anomalous behaviour that is caused by mis-configured and malicious devices.

2.1 ARP Broadcast Distribution

The data presented in Figure 1 shows that the number of devices broadcasting ARP requests at any given moment is fairly low. There are, however, 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.

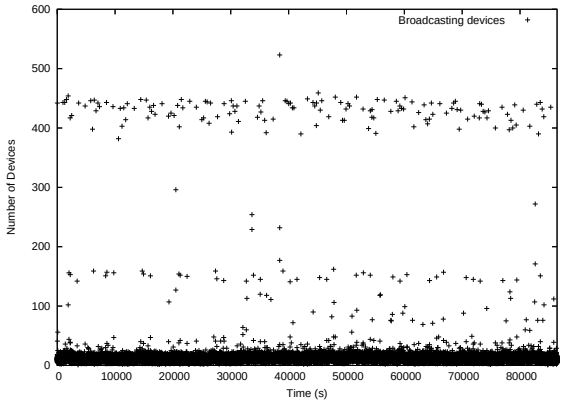

Figure 1: The number of devices broadcasting ARP requests at a given second over one day, taken from $N3$

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 behaviour 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 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.

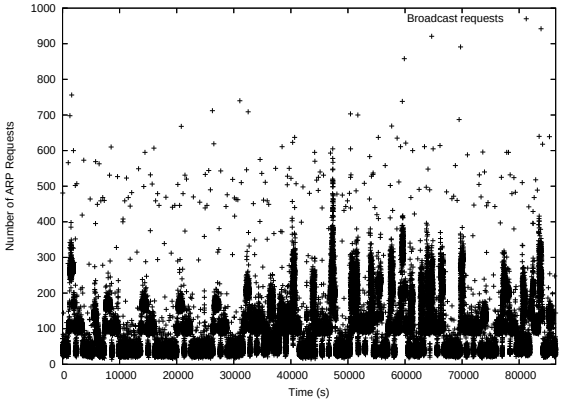

Figure 2: The number of ARP request broadcast at a given second over one day, taken from $N3$

The latter class of spikes can be explained by two forms of behaviour. The first form of behaviour 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 behaviour causing 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 ($N2$) 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 network[1]. 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 [10] looking for machines vulnerable to infection (see Section 2.2). Today, no attempt is made to limit the effects scanning worms have on the performance of Ethernet networks.

[1]This only applies to networks where the hosts have public IP addresses and are not, for example, behind NAT.

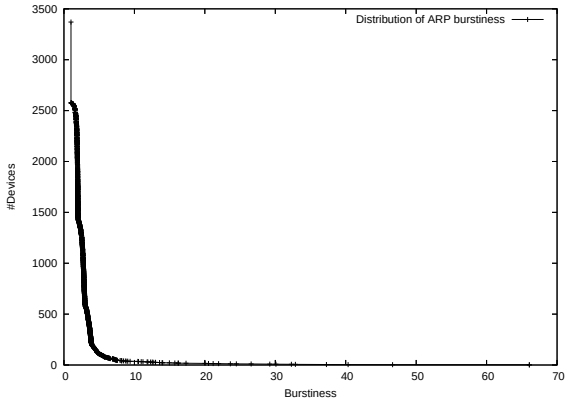

Figure 3: Burstiness characteristics of ARP traffic

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, 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, 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 Section 2.2) 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.

From the description of the ARP protocol and the analysis presented in this section, it should be clear that a normal ARP request rate is one whose sustained rate is well below 1 request/second (occasionally exhibiting very short "bursts" at around 1 request/second).

2.2 ARP Traffic Patterns

The ARP data analysis reveals an interesting characteristic of the hosts on the network – their traffic patterns. The data, illustrated in Figure 4 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 and 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. In other words, address mappings are removed from the ARP table prematurely to accommodate for new ARP table entries. Shortly after removing the 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 $N3$ show that about 81% of requests sent by its gateway are for unbound addresses, which suggests a significant number of external scans.

Furthermore, 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 *unicasting* an ARP request. If the target does not respond, another request is *broadcast* 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 broadcast by gateways and servers.

Figure 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 behaviour 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.

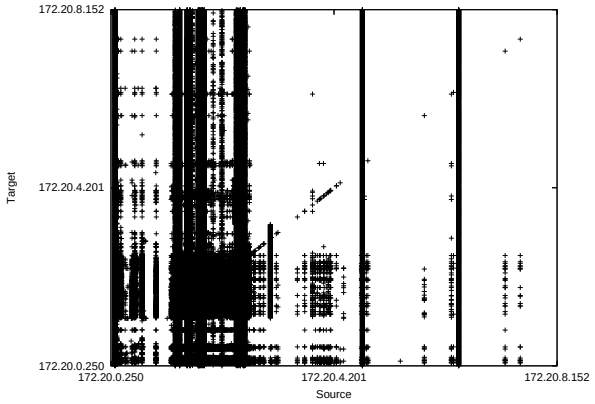

Figure 4: Traffic patterns showing the relationship between ARP sources and targets. This graph plots data from network $N1$ with the axes representing device identifiers.

2.3 The Effect of Malicious and Mis-configured Devices

As mentioned earlier, one of the data sets used in our ARP analysis was taken during a period where the some 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 behaviour 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. Firstly, a single firewall breach allows a worm to infect the whole protected subnet. Secondly, machines on a local area network are likely to be managed by the same individuals and be fairly homogeneous, therefore sharing the same vulnerabilities.

Local probes by a worm result 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 ($N2$) became 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 behaviour is demonstrated in Figure 5. One particular day saw about 120 separate devices per-

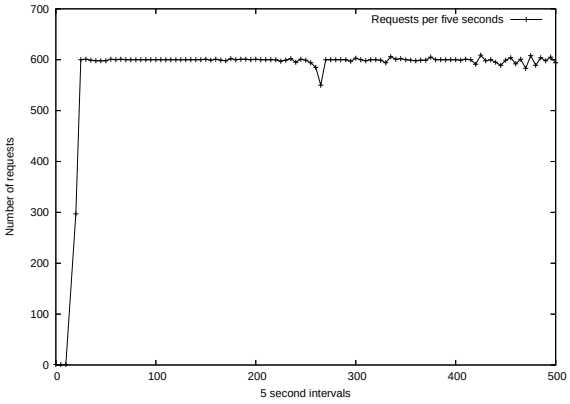

Figure 5: The increase in request rate for hosts infected with a worm scanning the local network for vulnerable machines.

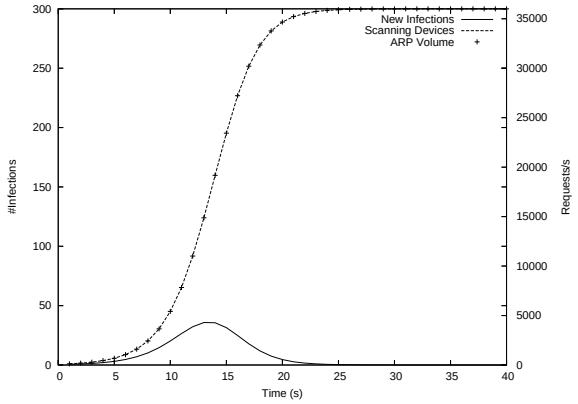

Figure 6: The infection dissemination and ARP volume caused by infected devices for $0 <= i <= 40$

forming 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.

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 [1] 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 shown in Equation 1.

$$m_i = \sum_{j=0}^{i} n_j \qquad s_{i+1} = \sum_{k=i-\frac{T}{s}}^{i} n_k$$

$$n_{i+1} = [N - m_i][1 - (1 - \frac{1}{T})^{rs_{i+1}}] \tag{1}$$

In Equation 1, 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 the request behaviour seen on the residential network, assumes a single initial infection ($n_0 = 0$), a class B 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 6, shows the worm 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, it takes the worm approximately 140 seconds to infect all the vulnerable hosts. 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 promotes the case for smaller subnets.

ARP requests are minimum-sized Ethernet frames of 64 bytes. As these ARP requests need to be broadcast on the network, an 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 requests/second, the load on the switch is $O(mnr)$ requests. Consequently, a few infected hosts can quickly consume a large amount of the forwarding capacity of a switch. In the above example, there are approximately 300 devices that are each broadcasting 120 request/second. This would cause a 24-port switch to broadcast about 828000 requests/second, using the slow path.

Equally as serious, is the effect that worms have 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. An estimate of 6 μs per interrupt, context switch and packet copy, and 36000 requests/second, results in a workstation wasting about 22% of its CPU cycles[2]. Interrupt coalescing, where implemented, will reduce the interrupt overhead somewhat. Additionally, the requests take up bandwidth on the links to end-stations – 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 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

[2]The numbers are based on [11] and analysis of Linux data path performance.

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 utilisation is low, only a small set of requests result in a reply. This behaviour 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.

3. MANAGING ANOMALOUS BEHAVIOUR

As seen in Section 2, 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 behaviour 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 behaviour. The algorithm can be seen in Algorithm 1 and 2. The strategy we propose can be described as follows.

Data: v - a switch wide moving average request rate. Initialized to 0.
Data: *table* - a table containing devices seen and timestamp of last ARP request seen.

```
1  begin
2      foreach incoming requests req do
3          s ⟵ 1 / (now − last request arrival)
4          v ⟵ (1 − α) × v + α × s
5          if v >= threshold then
6              f = threshold / size-of(table)
7              if allow-request (req, f) then
               forward-request (req)
8              add (table, req)
9          end
10         else 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 = get-update-timestamp (table, sender-of (req))
3      r = 1 / (now − l)
4      p = max(0, (r − f) / r)
5      return throw-coin (p)
6  end
```

Algorithm 2: Allow-Request Procedure

The maximum amount of resources allocated to ARP broadcast traffic forwarding is configured on the switch. This quantity could be specified as a number of requests per second the switch should broadcast per port, multiplied by the number of active ports, for example. If the observed rate of aggregated ARP traffic at the switch becomes greater than the desired forwarding rate, a scheme is invoked to manage (and reduce) the ARP traffic forwarded at the switch.

In this scheme, each active host (generating ARP requests) that is observed at a switch is allocated a share of the available resources allowed for request forwarding. If a host exceeds this share, its ARP requests are probabilistically dropped. This dropping is carried out in such a way that hosts that attempt to consume larger amounts of a switch's resources will suffer an increased probability of having their requests dropped. This scheme continues until the aggregated ARP request rate at the switch falls below the maximum forwarding rate, when the scheme is reset and "unmanaged" ARP forwarding resumes (ARP requests are no longer deliberately dropped).

To be more specific, the probability p_i that a host 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 2, where the value r_i is the instantaneous request rate for host i, calculated using Equation 3. Here, l_i represents the time at which the previous request by host i was received. The instantaneous rate and drop probability are calculated in Algorithm 2. The instantaneous request rate is used because normal ARP traffic from a host, as indicated in Section 2, is not transmitted in bursts.

$$p_i = max(0, \frac{r_i - f}{r_i}) \tag{2}$$

$$r_i = \frac{1}{now - l_i} \tag{3}$$

$$f = \frac{t}{n} \tag{4}$$

$$v = (1 - \alpha) \times v + \alpha \times s, \quad with \quad \alpha > 0 \tag{5}$$

$$s = \frac{1}{now - \max_i l_i} \tag{6}$$

As mentioned earlier, the f value represents the share of the switch's ARP request forwarding resources that are allocated to an individual active host. This value is determined using Equation 4, where the value t (given in requests/sec) represents the switch-wide maximum desired ARP request forwarding rate (threshold), and n is the number of hosts seen sending ARP requests on the switch's ports since the dropping scheme was invoked.

The devices known to the dropping scheme are forgotten and not used to calculate the share of the switch's available resource after a timeout period. This is so that the share of the available resources (f) is not effected by hosts 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 devices in this way has two consequences – one is to decrease the likelihood of dropping requests sent by hosts 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 isn't a 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.

The average switch rate (v) is computed as a moving weighted average of switch wide request rate samples, as in Equation 5, because the combination of non-bursty ARP traffic streams from hosts can result in some burstiness at the switch. 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.

The resources required to implement this relatively simple scheme on a switch should not be detrimentally high. The most significant amount of additional state to be maintained is a time-stamp of the last request for a given host – a small addition to a switch's forwarding table. In terms of the additional computation required, a switch needs to maintain the average ARP request rate across the switch and perform the operations described, which we believe for out-of-band traffic such as ARP is not prohibitively expensive.

4. EVALUATION

To give an indication of the effectiveness of the approach described in Section 3 for managing anomalous ARP behaviour, we carried out a series of simulations.

A network switch was modelled and extended with the probabilistic dropping strategy. The value of α, which is used to calculate the average rate at the switch (see Equation 5) was set to 0.2, and entries in the table of time-stamps were set to timeout after 30 seconds. A number of different threshold t values were used.

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 2. 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.

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

Table 2: Network information for traces used in simulations

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 2). Figures 7, 8, and 9, plot these values[3] for networks N2, N3, and N3 plus the synthesised scan traffic, respectively. The threshold value t was set to 128 requests/second.

These figures show that the hosts with higher instantaneous request rates are the most susceptible to having their ARP requests dropped. Differences in the percentage of requests dropped for hosts with similar request rates can also be observed. For instance, hosts shown in Figure 7 with a maximum request rate of 1000 requests/second have between 2% to 100% of their requests dropped. There are a number of causes of this. For example, over an extended 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. Conversely, a host may generate a burst of requests while the scheme is invoked (and be otherwise silent) and have a significant percentage of its requests dropped. But, these results demonstrate that our proposed scheme is indeed biased towards dropping ARP requests from high rate request "streams" – the required behaviour.

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 categorisation are indeed anomalous, many anomalous requests could still be categorised as normal (for example, abnormally high request rates towards bound addresses).

Figures 10, 11, and 12 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 10), 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.

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

[3]Note that the maximum instantaneous rates shown are never greater than 1000 requests/second because our algorithm was implemented at a millisecond time resolution. Better time resolution seems superfluous, as a host generating ARP requests at a higher rate should certainly be deemed malicious.

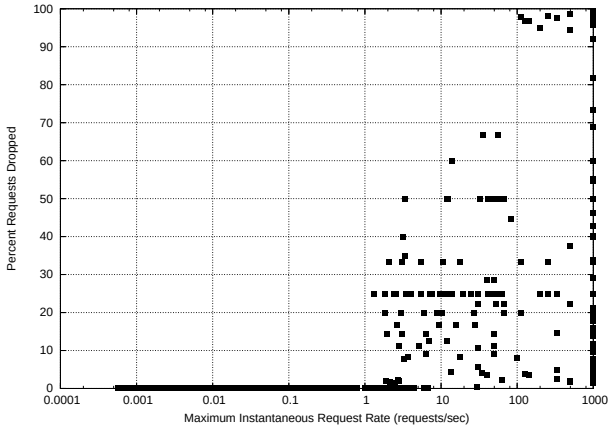

Figure 7: Percentage of requests dropped against the maximum instantaneous request rate for each host on network $N2$

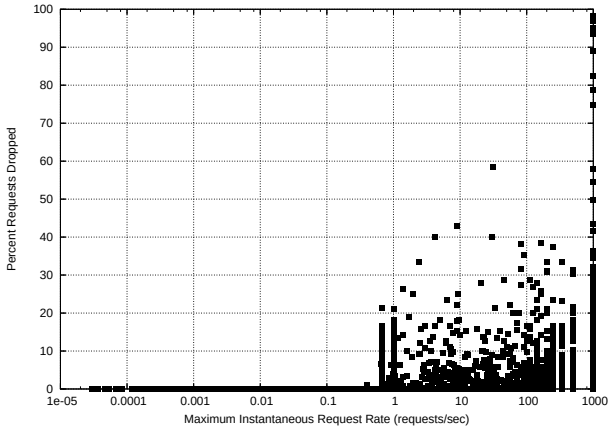

Figure 8: Percentage of requests dropped against the maximum instantaneous request rate for each host on network $N3$

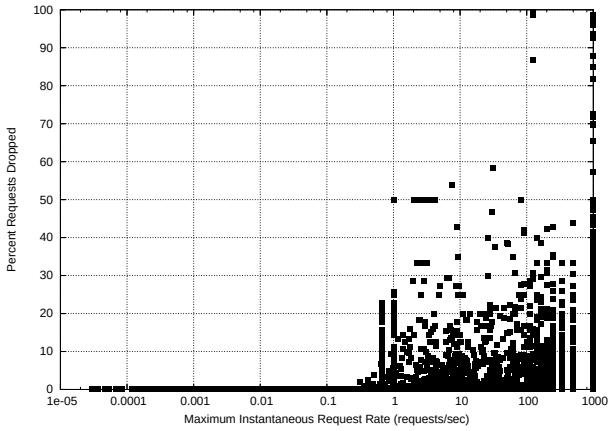

Figure 9: Percentage of requests dropped against the maximum instantaneous request rate for each host on network $N3$ and 25 synthesised network scans

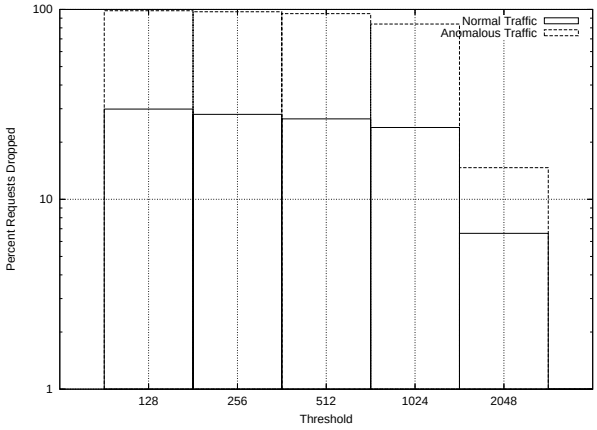

Figure 10: Percentage of the normal and anomalous ARP requests dropped for network $N2$ with different threshold values.

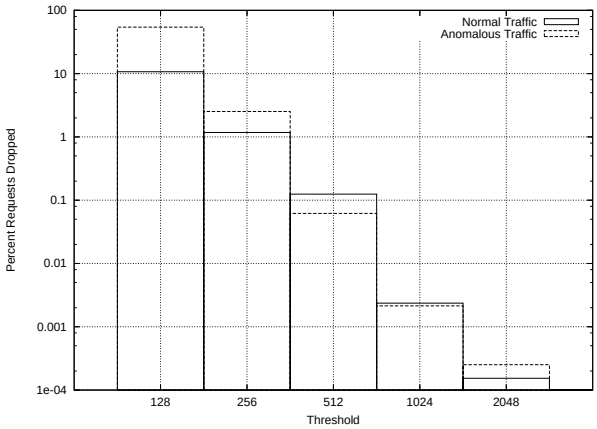

Figure 11: Percentage of the normal and anomalous ARP requests dropped for network $N3$ with different threshold values.

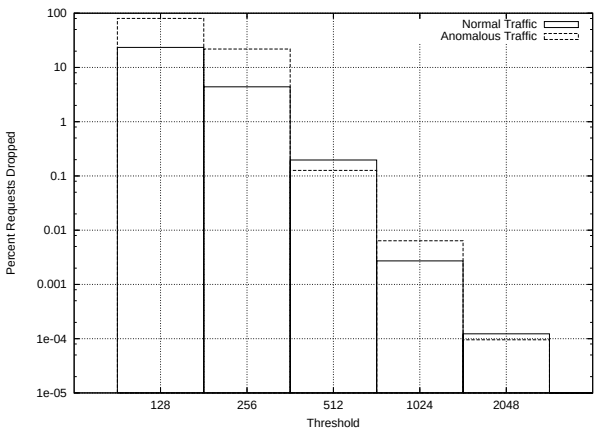

Figure 12: Percentage of the normal and anomalous ARP requests dropped for network $N3$ with synthesised scans with different threshold values.

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 router gateway, as this request stream can contain a significant amount of anomalous requests (see Section 2.2). However, in Sections 5 and 6, 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 request 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 11 and 12, 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 11), 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 10, 11, and 12. 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 13 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

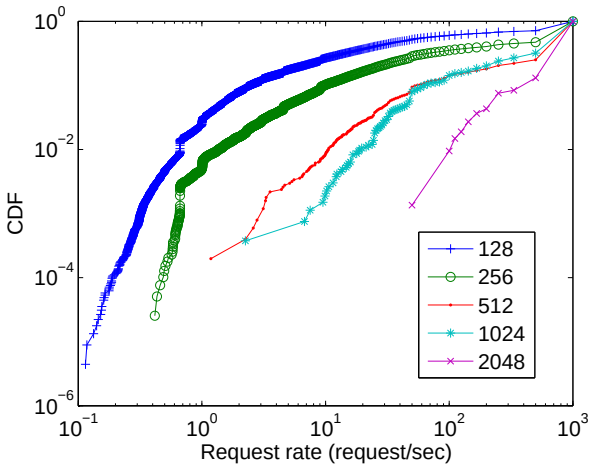

Figure 13: CDF of the number of drop requests across all three networks for various threshold values.

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.

As mentioned in Section 1.1, 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 14, 15, and 16 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 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.

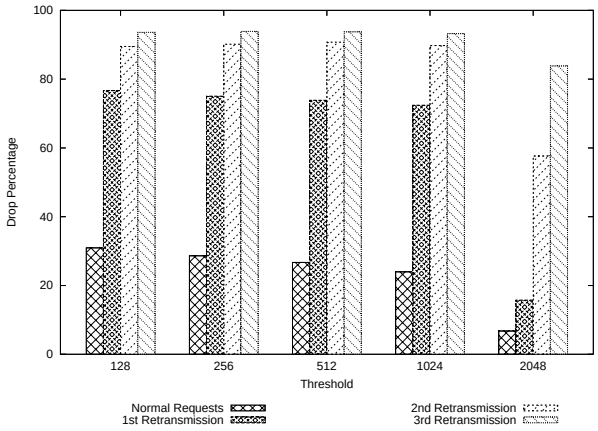

Figure 14: Percentage of the normal requests and retransmissions that are dropped for network $N2$

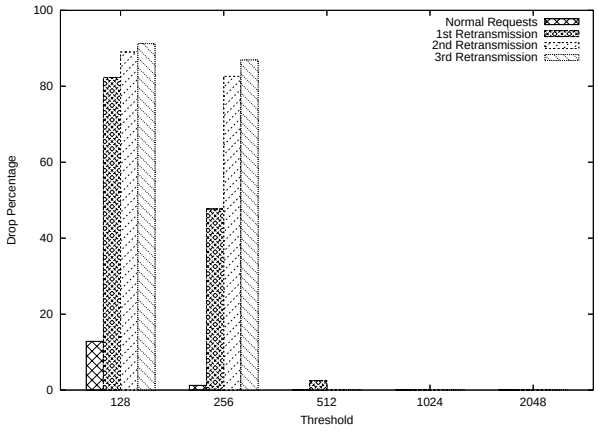

Figure 15: Percentage of the normal requests and retransmissions that are dropped for network $N3$

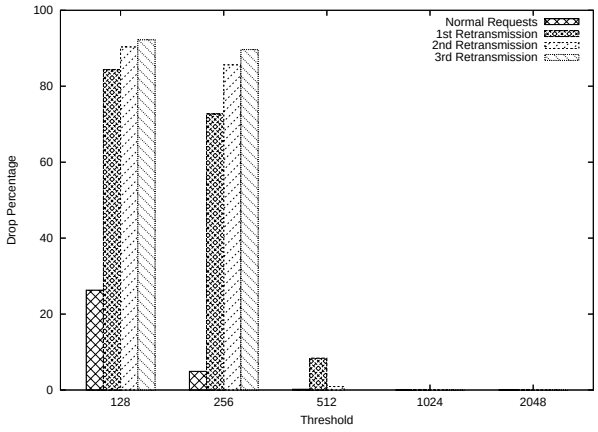

Figure 16: Percentage of the normal requests and retransmissions that are dropped for network $N3$ with synthesised scans

5. RELATED WORK

Considerable research effort has gone into understanding the scalability of Ethernet-based local area networks. Some claim that the spanning-tree construction carried out on Ethernet switches is the main bottleneck on Ethernet scalability. This is because the use of a single spanning-tree prevents the utilisation of redundant network resources and decreases fault-tolerance. This fact in turn prevents load-balancing and increases end-to-end latency. These are well-known effects of the use of spanning-trees. In [8], proposals are presented for a VLAN-based architecture to construct multiple spanning-trees. Additionally, in [4] an approach is provided for mapping several spanning-trees to a reduced number of VLANs. The use of VLANs could also be used to manage the broadcast domains of a local area network. The only drawback associated with this approach is that VLANs require additional configuration.

In [5], 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. It is unclear how this would work with existing infrastructure. Our proposal addresses the potential threat caused by ARP in light of anomalous behaviour, and can be used to control its potential impact. A solution similar to ours could be used in a more general context to reduce the effect of other broadcast traffic on Ethernet networks in addition to ARP, therefore obviating the need for the proposed directory services.

The scheme presented in Section 3 is similar in nature to research carried out on Internet worm containment, in particular network-based approaches to worm containment. Snort [7] and Network Security Monitor [3] are two examples of network-based worm containment approaches. While it is not a direct goal of ours to detect and contain Internet worms, our scheme could be used to reduce their rate of proliferation. As such, we believe the approach presented here is complimentary to such schemes.

Black hole routers/sink holes [2] are an elegant solution to deal with external scans (i.e. scans originating from outside the LAN and causing the network gateway to generate excessive ARP request broadcasts to unbound addresses). 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. Because the gateway will identify the black hole router as its next hop to any unbound address, all traffic to such addresses can be dealt with at the gateway *without* triggering ARP request broadcasts, since, for that traffic, the only ARP binding needed at the gateway is that of the black hole router. 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.

6. CONCLUSIONS

In this paper, we have studied the behaviour of ARP traffic. We have shown that although ARP raises no cause for concern for even very large Ethernet networks under normal conditions, the presence of malicious software and external scans can be seen as a problem.

We have presented a simple, yet very effective scheme to control the amount of ARP traffic present in the network. Our scheme efficiently isolates 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, server or gateway router generating anomalous ARP request patterns.

Also, because of externally generated scans, a gateway router often appears to generate a substantial amount of anomalous ARP traffic. Filtering out the normal ARP traffic from the malicious one emanating from a gateway is one of the greatest challenges faced by our proposed scheme. Although our scheme does not guarantee that "legitimate" ARP traffic from a gateway (or from any host generating a mix of malicious and legitimate ARP requests) will not be dropped, it should be noted that widely used techniques can help alleviate the problem: static ARP table entries for servers (on the gateway) or defining a subnet where only servers reside (a.k.a, a demilitarized zone) can ensure that communications to servers are never hindered by our proposed scheme. Better still, the use of black hole routers or sinkholes within a network, or whenever appropriate the use of firewalling rules limiting inbound traffic, can efficiently filter out most of the externally generated malicious traffic.

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 an approximate 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, our scheme discourages 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 paper 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, such as wireless multi-hop networks.

7. ACKNOWLEDGEMENTS

The work presented in this paper has been partially supported by the British Department of Trade and Industry (DTI) under the Ubicare Center-lead ANS Project, and also partially supported by the EU E-NEXT Network of Excellence.

8. REFERENCES

[1] Z. Chen, L. Gao, and K. Kwiat. Modeling the Spread of Active Worms. In *IEEE INFOCOM 2003*, 2003.

[2] B. Greene and D. McPherson. Sink Holes: A Swiss Army Knife ISP Security Tool. http://www.arbornetworks.com/downloads/research36/Sinkhole_Tutorial_June03.pdf.

[3] L. T. Heberleid, G. Dias, B. Mukerjeeand Levitt K, J. Wood, and D. Wolber. A Network Security Monitor. In *Proceedings of the IEEE Symposium on Research in Privacy*, 1990.

[4] IEEE. IEEE 802.1s Multiple Spanning Trees. http://www.ieee802.org, 2002.

[5] A. Myers, E. Ng, and H. Zhang. Rethinking the Service: Scaling Ethernet to a Million Nodes. In *ACM SIGCOMM HotNets 2004*, San Diego, CA, USA, November 2004.

[6] D. C. Plummer. 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, November 1982.

[7] M. Roesch. Snort: Lightweight Intrusion Detection for Networks. In *Proceedings of the 13th Conference on System Administration*, November 1999.

[8] S. Sharma, K. Gopalan, S. Nanda, and T. Chiueh. Viking: A Multi-Spanningtree Ethernet Architecture for Metropolitan Area and Cluster networks. In *IEEE INFOCOM 2004*, Hong Kong, China, March 2004.

[9] Symantec. W32.Blaster Worm. http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.worm.html.

[10] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham. A Taxonomy of Computer Worms. In *In the First ACM Workshop on Rapid Malcode (WORM)*, Washington DC, USA, October 2003.

[11] M. Zec, M. Mikuc, and M. Zagar. Estimating the Impact of Interrupts Coalescing Delays on Steady State TCP Throughput. In *Proceedings of the 10th SoftCOM 2002*, 2002.