Efficient and Low-Cost Hardware Defense Against DNS Amplification Attacks

Changhua Sun, Bin Liu and Lei Shi
Tsinghua National Laboratory for Information Science and Technology
Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, China
{sch04,shijim}@mails.tsinghua.edu.cn, liub@tsinghua.edu.cn

Abstract—**DNS amplification attacks utilize IP address spoofing and large numbers of open recursive DNS servers to perform the bandwidth consumption attack. During an attack, it ceaselessly fabricates DNS queries to the exploited open recursive DNS servers, and all the responses, often with larger size than the query messages, are reflected to the single victim due to the source IP address spoofing. While it is difficult to defend against this attack from the root causes by eliminating the open recursive DNS servers and IP spoofing for the whole Internet, in this paper, we take a different methodology to defend against it at the leaf router of victim's ISP or organization. We propose an efficient and low-cost hardware approach to first detect the DNS amplification attack accurately and responsively. Once the attack is confirmed, our approach is then activated to filter out all the illegitimate DNS responses by using a two-Bloom filter solution. We demonstrate that the memory cost of our approach is feasible for the hardware implementation even up to the OC-768 link. Through trace-driven simulations, it is shown that our approach is effective in both the detecting and filtering phases.**

I. Introduction

Distributed Denial-of-Service (DDoS) attacks [1] are still a serious threat to the Internet. By taking advantage of the lack of authenticity in the IP protocol, destination oriented routing, and core-stateless nature of the Internet, DDoS attacks can easily throw out Internet services like Web and Email. The Domain Name Service (DNS) [2], Internet's address book, mainly translates alphabetic domain names (like URL, Email address) to numerical IP addresses. As nearly all the Internet services rely on DNS, DDoS attacks against or using DNS servers are much more damaging than the other cases. On October 21 2002, a coordinated DDoS attack flooded all root servers simultaneously and brought serious response delay for some root servers [3]. There are many other similar attacks launched during 2004-2007 [4], [5]. In February 2006, ICANN SSAC reported several DNS amplification attacks [6], which used massive DNS servers to flood other victims.

These attacks against or related to DNS servers are classified into two types. One is to directly flood DNS servers by sending a large number of DNS requests or other useless traffic. Since the DNS servers cannot easily distinguish the legitimate requests from the attack traffic, they would simply accept both of them and send the responses. The effective and deployable defense against this attack is to over-provision the network capacity and numbers of servers, like the anycast technology [7], which is proven to be effective in the Feb-6-2007 DDoS attacks flooding root servers [8], [9].

This work is partly supported by NSFC (60573121, 60625201), China 973 program (2007CB310701), the Specialized Research Fund for the Doctoral Program of Higher Education of China (20060003058), Tsinghua Basic Research Foundation (JCpy2005054), 863 high-tech (2007AA01Z216).

The other one is to utilize the exploited open recursive DNS servers to amplify the attack traffic to a certain target victim. The open recursive servers are those that accept arbitrary DNS queries from any source and send the final response answers to the queries. Since the source IP address in the queries are spoofed as the victim's and the elaborate DNS query could be much smaller than its associated response in size, the attack traffic is actually amplified while reflected to the victim server, which is called the DNS amplification attacks [10]. In an extreme case, even a mild attack traffic of 1Gbps [10] could stuff the fastest link currently available in the Internet, and the victim server is doomed to denial-of-service. It is very difficult to deal with this kind of attack.

In this paper, we propose an efficient and low-cost hardware approach to defend against the DNS amplification attacks. We exploit the semantic feature of DNS protocol: one-to-one mapping of the DNS requests and responses. We also eliminate the need to maintain the mapping relationship and avoid the storage complexity to make the approach more practical. Our approach includes two phases: *Detection* and *Filter*. The detection phase is to detect the DNS amplification attacks with high accuracy and short response time, more importantly with little cost, as we just need several counters. After we detect that the amplification attacks are serious enough to exceed a predefined threshold, the filter scheme starts to work, which we call the filter phase. The main task of the filter scheme is to distinguish the legitimate DNS responses from the attack traffic with an affordable computation and memory cost. To handle that, we propose a two-Bloom filter [11] solution. In our filter scheme, the memory needed for the OC-192 (SONET, 10Gbps) link is 4.36 MB. Using the state-of-the-art SRAM, we only need one 9-MB SRAM for the OC-192 link and two SRAMs even for the OC-768 (40Gbps) link, making it feasible and attractive for the hardware implementation. Through trace-driven simulations, it is demonstrated that our approach is very robust and effective in both the detecting and filtering phases. Furthermore, it can be deployed at the ISP's leaf router to protect its high-end customers, which actually provides enough incentive for the deployment.

The rest of the paper is organized as follows. Section II describes the background and related work. Section III proposes our hardware approach to defend against the DNS amplification attacks in detail. Section IV presents the performance evaluation results through trace-driven simulations. Finally in section V, we conclude the paper.

II. Background and Related Work

First, we show a typical scenario of DNS amplification attacks. An attacker controls a Botnet [12] comprised of many zombie computers and continuously pushes thousands of DNS queries to the exploited open recursive DNS servers, while spoofing the victim server's IP as the source address. In such attacks, each DNS request encapsulated with a 60-byte datagram could be answered with a much larger 4000-byte response due to the effect of EDNS [13], hence reaching a bandwidth amplifying factor of more than 60 [10].

The root causes for the DNS amplification attacks are threefold: 1) Many DNS name servers are recursive. A recent survey shows over 75% of domain name servers allow recursive name service to arbitrary queries [14]. 2) The Internet architecture itself cannot prohibit spoofing. Though ingress address filtering [15] or unicast reverse path forwarding checks [16] can prevent spoofing in practice, these filtering techniques are limited by multi-homing, route asymmetry, filter list maintenance and little incentive in real networks. Spoofing still exists according to [17]. More specific, since DNS mainly uses UDP protocol, it is much easier to spoof source IP address as UDP is connectionless and does not need to fulfill the handshake used in the connection-oriented protocol like TCP. 3) In most cases, smaller DNS queries can generate larger DNS response messages in length. Normally, the length of an UDP response message is limited to 512 bytes [18], but due to the effect of EDNS [13], there is no such a limitation for the UDP response message.

ICANN SSAC has given a reference solution to deal with the DNS amplification attacks [6], which utilizes source IP address verification, advanced security for open recursion on name servers from external sources, as well as the security in DNS servers operations. These recommendations are expected to thoroughly eliminate the DNS amplification attacks when widely deployed. However, ISPs actually have a low incentive to deploy such a solution since ISPs who deploy it would prevent others not themselves from suffering the DNS amplification attacks. Therefore, in a short term, it will be quite helpful and valuable to design an efficient and low-cost approach to defend against the DNS amplification attacks in an incentive-driven manner.

There is little research work towards the DNS amplification attacks. DNS-Guard [19] is proposed to detect the spoofing DNS requests. It involves several policies that generate some form of cookies for a DNS server to implement origin authentication. However, it can only verify the requests between DNS servers. The requests from general DNS resolver clients, which are the main attack tools in the amplification attacks, cannot be verified. Kambourakis et al. [20] proposed a fair solution to the DNS amplification attacks based on the one-to-one mapping of the DNS requests and responses. But the database size would increase rapidly in cases of high traffic rate, making their approach unscalable for practical use.

III. Detection and Filter Schemes

A. Detection Scheme

Our detection scheme utilizes the semantic feature of DNS protocol. Usually, one DNS request message would generate one DNS response message (A large response message would break up to several packets due to the IP fragment when its length exceeds the MTU). Our detection and filter scheme would monitor the outgoing DNS requests (from the inner of ISP or organization to the Internet) and the incoming DNS responses. Normally, one incoming response would match one outgoing request. During the DNS amplification attacks, if the Botnet and exploited open recursive DNS servers are outside of the ISP or organization, the requests from the Botnet to the recursive DNS servers would not be discovered by our scheme. However, the attack DNS responses from the DNS servers to the victim in the ISP or organization would be inevitably captured. Therefore, the number of the responses would more than that of the DNS requests during the attack. Notably, our detection and filter schemes both assume the attack computers (or Botnet) and the recursive DNS servers are outside of the protected ISP or organization.

First, we need to identify the DNS request and response packet efficiently. Since the DNS amplification attacks exploit IP spoofing and it is difficult to spoof IP address in TCP for the three-hand handshake, we only need to monitor the DNS packets using UDP. The DNS request or response packet should satisfy the following requirements: 1) IP packet without fragmentation or it is the first packet of fragmentation segments. In both cases, the IP fragmentation offset should be zero; 2) DNS packet (UDP and source or destination port is 53); 3) For outgoing DNS request packet, destination port is 53 and response flag is zero; while for incoming DNS response packet, source port is 53 and response flag is one.

Next, in each time interval δ, we count the number of the DNS requests q and the DNS responses p. As the matched response is some time later than the request, there would be some difference between q and p. From a statistic view, the behavior between the difference in normal condition should be different from that in DNS amplification attacks. We use a low-pass filter to detection the attack and define *detection value* S_δ as follows.

$$S_\delta = \alpha \cdot S_{\delta-1} + (1-\alpha) \cdot (p-q)/\max(q,1). \tag{1}$$

where $S_0 = 0$ and α indicates the portion of history in detection value and how fast the difference in this time interval to affect the detection value. Normally, S_δ is almost zero. During the DNS amplification attacks, there are more responses than requests and S_δ would be greater than zero. A threshold th can be given to measure the serious of the attacks, and trigger the working of the filter scheme. Roughly,

```
Initialization;
$\delta = 2$; $t_\delta = t + \delta$; $Turn = 1$;
Clear $BF_A$ and $BF_B$;
while 1 do
    update $t$;
    if DNS request then
        if $Turn$ then Insert it to $BF_A$;
        else Insert it to $BF_B$;
    end
    if DNS response then
        if not in $BF_A$ and not in $BF_B$ then drop it;
    end
    if $t \geq t_\delta$ then
        $t_\delta = t_\delta + \delta$;
        if $Turn$ then $Turn = 0$; Clear $BF_B$;
        else $Turn = 1$; Clear $BF_A$;
    end
end
```

Fig. 1. The algorithm of the filter scheme

th represents the number of response is $th+1$ times of request in average and can be predefined according to the protected ISP's capability to withstand the attacks. α is used to smooth the current detection value by the history value. We choose α as 0.4 in (1). The memory cost of our detection scheme is several counters, making it very attractive for the hardware implementation.

The detection scheme has no false positive but may have false negative as we report the attack according to the threshold th. This is acceptable if the attacks are not strong enough to bring any trouble or disaster to the victim.

B. Filter Scheme

Our filter scheme is to identify whether the DNS responses are legitimate or not. Our scheme would discover the one-to-one mapping between the outgoing DNS request and the incoming response. The main idea of the filter scheme is to efficiently store the outgoing requests. Since the filter scheme only starts to work when detecting serious attacks, if the incoming response cannot find a matched request, it can be classified as the attack response. To reduce the memory needed to store the requests, we introduce a two-Bloom filter [11] solution. Suppose the maximum time interval between the outgoing DNS request and its corresponding incoming DNS response, which are seen by our scheme, is δ. The two Bloom filters would alternately store the requests in each δ. At the beginning of each turn, the Bloom filter would be cleared. Therefore, any request would stay in the Bloom filters at least δ, at most 2δ. When receiving one incoming response, we can check whether its corresponding request is in either Bloom filter or not to decide if the response is legitimate.

A Bloom filter [11] is a simple space-efficient data structure for representing a set in order to support membership queries. To represent a set $V = \{a_1, a_2, ..., a_n\}$ of n elements, an array of m bits, which composes the Bloom filter, is initially set to all 0. k independent hash functions $h_1, h_2, ..., h_k$, hash value of each with range $\{0, 1, ..., m-1\}$, are used. For each element $a \in V$, the bits $h_i(a)$ are set to 1 for $1 \leq i \leq k$. A location can be set to 1 multiple times, but only the first change has an effect. If we want to query element b whether in V, we check whether all bits $h_i(b)$ are set to 1. If not, certainly b is not in V. Otherwise, b is assumed in V with some wrong probability. Because some locations of $h_i(b)$ may be set to 1 by other elements. The wrong probability is referred to as the *false positive* rate p_f, and it suggests that an element a is in V even though it is not. If $m > kn$, p_f is approximately $(1 - e^{-kn/m})^k$. By optimizing the number of hash functions with minimized false positive rate, we get:

$$p_f = (0.5)^k = (0.5)^{\ln 2 \cdot (m/n)}. \qquad (2)$$

The algorithm of the filter scheme is shown in Fig. 1 (t is the current time). One Bloom filter, BF_A would store *4-tuple* (source IP, destination IP, source port, DNS transaction ID[1]) of the requests during $[t, t+\delta)$, and the other, BF_B , would store the requests during $[t+\delta, t+2\delta)$. Next, BF_A would be reset (all bits are set to 0) and store $[t+2\delta, t+3\delta)$, and then BF_B would be reset and store $[t+3\delta, t+4\delta)$. Therefore, the 4-tuple of any outgoing DNS request would stay in the Bloom filters at least δ, at most 2δ.

When an outgoing DNS request is received, we extract its 4-tuple as an item and insert the item into the first or second Bloom filter. If an incoming DNS response is received, we extract its 4-tuple (destination IP, source IP, destination port, DNS transaction ID)[2] as an item and query this item from the two Bloom filters. If the item is in either Bloom filter, the response is legitimate with high probability. Otherwise, the response is illegitimate. We can drop it during the DNS amplification attacks. As the existence of false positive, some illegitimate responses may be recognized as the legitimate ones. This just means we may pass a few attack responses and would not affect the filter scheme. In addition, we can control the false positive as low as acceptable.

Reducing the memory needed by the filter scheme is important, especially in the implementation of high speed links. From (2), we can decide how many hash functions needed for an acceptable false positive. Suppose the maximum DNS request packet rate of the link is r pps (packet per second), the number of elements n supported by one Bloom filter in time interval δ is $n = r\delta$. From (2), the memory bits needed by the scheme are as follows.

$$M = 2m = 2nk/\ln 2 = 2r\delta k/\ln 2. \qquad (3)$$

r is related to the packet length, the traffic proportion of DNS request packets, which can be computed according to normal traffic condition since they are always legitimate, and overhead of the link [21]. Traffic measurements in [22], [23] collecting real traces from Sprint IP backbone show that less than 10%

[1]DNS transaction ID is the identification of DNS query and would be included in the corresponding response.

[2]The 4-tuple of the response matches with that of the corresponding request.

TABLE I
MEMORY NEEDED BY THE FILTER SCHEME

Link	Link Rate (Gbps)	rMpps	M(MB)
GE	1	0.116	0.34
OC-48	2.488	0.381	1.09
OC-192	9.953	1.523	4.36
10GE	10	1.158	3.32
OC-768	39.813	6.090	17.43

TABLE II
TRACE INFORMATION

Trace	Duration	IP	DNS	Request	DNS bytes
A2I	600s	8,627,358	67,162	54.46%	0.11%
I2A	600s	9,460,192	64,988	42.21%	0.14%
K2I	599s	43,229,001	390,551	50.12%	0.11%
I2K	599s	39,811,858	279,758	59.0%	0.076%

of the traffic is UDP and less than 3% of the traffic is DNS. Therefore, it is very safe to assume no more than 10% of the traffic is DNS requests. In addition, through real trace analysis, it is shown the average DNS request packet length is more than 70 bytes. So we choose 70 bytes as the average length to compute r. For hash functions to fulfill Bloom filter, we choose $k = 4$, and then $p_f = 0.0625$, which shows no more than 6.25% illegitimate responses are considered as legitimate. The time interval δ can be considered roughly similar to the round-trip time (RTT) in TCP protocol. It is shown that more than 90% of the TCP connections have an RTT that is less than 500ms [24]. It would be safe if $\delta = 2$ to make all legitimate responses recognized according to the Bloom filters. Thus, for $k = 4$ and $\delta = 2$, the memory needed by the filter scheme is $M = 3r$ bytes, and shown in Table I. The memory needed for the OC-192 link is about 4.36 MB, which just needs less than one 9-MB SRAM using the state-of-the-art SRAM technology [25]. Moreover, we only need two SRAMs even for the OC-768 link, making our filter scheme feasible for the hardware implementation. In addition, we choose H_3 as the hash functions [26] to implement the Bloom filter, as it is easy to be implemented by software or hardware.

The filter scheme can also use one counting Bloom filter [11], which can be easily used for inserting and deleting items as each entry in the Bloom filter is not a single bit but rather a small counter. However, the memory needed would be more than our two general Bloom filters solution.

IV. PERFORMANCE EVALUATION

First, we evaluate our detection scheme through trace-driven simulations. Though we utilize the semantic feature of DNS protocol, it is required to verify whether the difference between the number of the DNS requests and responses is constant or almost constant under normal conditions. We study several public available two-direction traces from NLANR [27] to evaluate the detection scheme. And we give the results of two examples whose trace information is shown in Table II. The link between IPLS and ATLA (`IPLS-ATLA-20040819-135000`) is OC-48 and IPLS and KSCY (`IPLS-KSCY-20040819-161000`) is OC-192. The

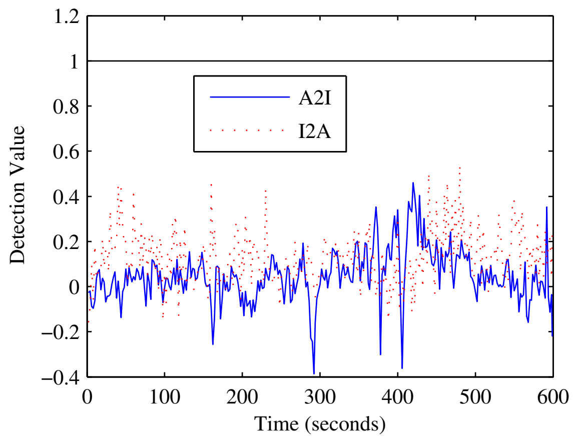

Fig. 2. The detection scheme on OC-48 traces.

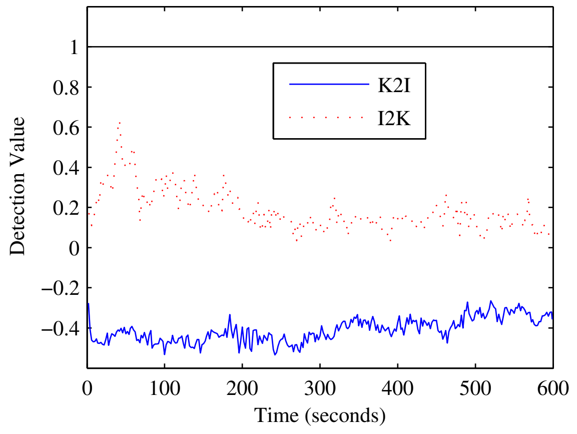

Fig. 3. The detection scheme on OC-192 traces.

requests in A2I and the responses in I2A are called the direction A2I in the following evaluations. So is the same with I2A, K2I and I2K directions.

In our detection scheme, we choose $th = 1$ as the threshold which would report the attack if the number of responses is roughly 2 times of the number of requests. The detection value of these traces is shown in Fig. 2 and Fig. 3. It is shown the detection values are all less than the th and the changes of the values are not much. This indicates the roughly constant between the number of the DNS requests and responses. As the traces are not collected at the ISP's leaf routers, they are actually collected at the points of backbone [27]. Due to multiple routing paths and asymmetric routing, the difference between the responses and requests is clearly shown in Fig. 2 and Fig. 3, especially for the IPLS-KSCY traces. If we collect the traces at the ISP's leaf routers, the detection values would be almost zero, which is shown in the following experiment.

Next, we collected real traces in the information center (holds several Web and Email servers) of our university. During several collected time points, we also launch self-attack against the information center. We just send the DNS

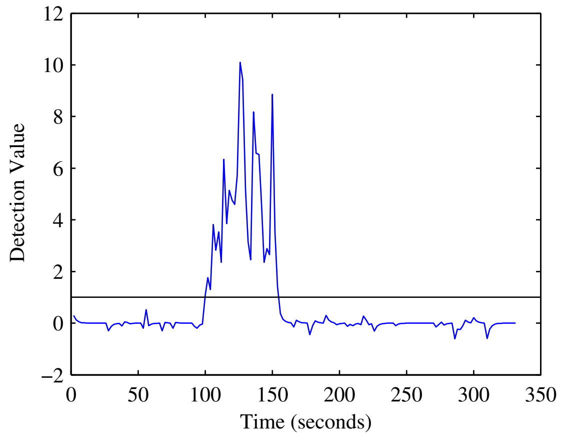

Fig. 4. Illustration of detecting the DNS amplification attack.

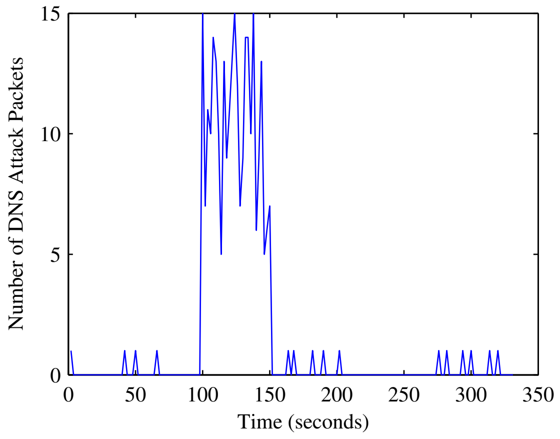

Fig. 5. Illustration of identifying attack traffic.

responses directly to the information center at the outside. We give one example result here. Fig. 4 shows the result using our detection scheme to detect the DNS amplification attacks. In this example, we launch the attack at about 100 seconds and stop it at about 150 seconds. It is shown our detection scheme can detect the attacks with high accuracy and short response time. In addition, during normal conditions, the detection values are close to zero, agreeing with our discussion in section III-A. Fig. 5 shows the attack packets identified by our filter scheme. These attack packets, expect a few useless miss-match responses, agree with our self-launch attack traffic.

V. Conclusion

In this paper, we propose an efficient and low-cost hardware approach to defend against the DNS amplification attacks at the leaf router of victim's ISP or organization. Our approach is to first detect the attack with high accuracy and short response time. Next we distinguish the legitimate DNS responses from the attack traffic using a two-Bloom filter solution. We have shown that the memory cost of our approach is feasible for the hardware implementation up to the OC-768 link. We also demonstrate through trace-driven simulations, that our approach is efficient and robust in both the detection and filter phases in dealing with the DNS amplification attacks. Our hardware approach can be deployed at the ISP's leaf routers to protect its own customers, hence brings incentive for the incremental deployment in the Internet.

References

[1] "CERT Advisory CA 1999-17, Denial-of-Service Tools," 1999. [Online]. Available: http://www.cert.org/advisories/CA-1999-17.html

[2] P. Mockapetris, "Domain names - concepts and facilities," RFC 1034, 1987. [Online]. Available: http://www.ietf.org/rfc/rfc1034.txt

[3] P. Vixie, G. Sneeringer, and M. Schleifer, "Events of 21-Oct-2002," 2002. [Online]. Available: http://d.root-servers.org/october21.txt

[4] R. Lemos and J. Hu, "'Zombie' PCs caused Web outage, Akamai says," 2004. [Online]. Available: http://www.news.com/Zombie-PCs-caused-Web-outage,-Akamai-says/2100-1038_3-5236403.html

[5] "Factsheet: Root server attack on 6 February 2007," ICANN, 2007. [Online]. Available: http://www.icann.org/announcements/factsheet-dns-attack-08mar07.pdf

[6] "SSAC Advisory SAC008 DNS Distributed Denial of Service(DDoS) Attacks," ICANN SSAC, 2006. [Online]. Available: http://www.icann.org/committees/security/dns-ddos-advisory-31mar06.pdf

[7] T. Hardie, "Distributing authoritative name servers via shared unicast addresses," RFC 3258, 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3258.txt

[8] S. Cheung, "Denial of service against the domain name system," *IEEE Security & Privacy Magazine*, vol. 4, no. 1, pp. 40–45, 2006.

[9] G. Lawton, "Stronger domain name system thwarts root-server attacks," *IEEE Computer*, vol. 40, no. 5, pp. 14–17, May 2007.

[10] R. Vaughn and G. Evron, "DNS amplification attacks," 2006. [Online]. Available: http://www.isotf.org/news/DNS-Amplification-Attacks.pdf

[11] A. Broder and M. Mitzenmacher, "Network applications of Bloom filters: a survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2004.

[12] "Know your enemy: tracking Botnets," 2005. [Online]. Available: http://honeynet.org/papers/bots/

[13] P. Vixie, "Extension mechanisms for DNS (EDNS0)," RFC 2671, 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2671.txt

[14] "The measurement factory DNS survey," 2005. [Online]. Available: http://dns.measurement-factory.com/surveys/sum1.html

[15] P. Ferguson and D. Senie, "Network ingress filtering: defeating denial of service attacks which employ IP source address spoofing," RFC 2827, 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2827.txt

[16] F. Baker and P. Savola, "Ingress filtering for multihomed networks," RFC 3704, 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3704.txt

[17] R. Beverly and S. Bauer, "The spoofer project: inferring the extent of source address filtering on the Internet," in *USENIX workshop on Steps to Reducing Unwanted Traffic on the Internet*, July 2005.

[18] P. Mockapetris, "Domain names - implementation and specification," RFC 1035, 1987. [Online]. Available: http://www.ietf.org/rfc/rfc1035.txt

[19] F. Guo, J. Chen, and T. Chiueh, "Spoof detection for preventing DoS attacks against DNS servers," in *IEEE ICDCS*, 2006, pp. 37–37.

[20] G. Kambourakis, T. Moschos, D. Geneiatakis, and S. Gritzalis, "A fair solution to DNS amplification attacks," in *International Workshop on Digital Forensics and Incident Analysis (WDFIA)*, 2007, pp. 38–47.

[21] P. Dykstra, "Protocol overhead," 2003. [Online]. Available: http://sd.wareonearth.com/~phil/net/overhead/

[22] C. Fraleigh *et al.*, "Packet-level traffic measurements from the Sprint IP backbone," *IEEE Network*, vol. 17, no. 6, pp. 6–16, 2003.

[23] "IP Data Analysis," Sprint academic research group. [Online]. Available: http://ipmon.sprint.com/packstat/packetoverview.php

[24] J. Hao and D. Constantinos, "Passive estimation of TCP round-trip times," *Comput. Commun. Rev.*, vol. 32, no. 3, pp. 75–88, 2002.

[25] "Cypress Semiconductor." [Online]. Available: http://www.cypress.com

[26] M. V. Ramakrishna, E. Fu, and E. Bahcekapili, "Efficient hardware hashing functions for high performance computers," *IEEE Trans. Comput.*, vol. 46, no. 12, pp. 1378–1381, 1997.

[27] "Abilene-V Trace Data," NLANR Project, 2004. [Online]. Available: http://pma.nlanr.net/Special/ipls5.html