Computer Networks 55 (2011) 1037–1051

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

WDA: A Web farm Distributed Denial Of Service attack attenuator

Ehud Doron [*], Avishai Wool

School of Electrical Engineering, Tel-Aviv University, Ramat-Aviv, Israel

ARTICLE INFO

Article history:
Received 9 June 2009
Received in revised form 2 May 2010
Accepted 4 May 2010
Available online 1 June 2010
Responsible Editor: Christos Douligeris

Keywords:
Distributed Denial Of Service
Network security

ABSTRACT

Distributed Denial Of Service (DDoS) attacks are familiar threats to Internet users for more than 10 years. Such attacks are carried out by a “bot-net”, an army of zombie hosts spread around the Internet, that overwhelm the bandwidth toward their victim Web server, by sending traffic upon command. This paper introduces WDA, a novel architecture to attenuate the DDoS attacker’s bandwidth. WDA is especially designed to protect Web farms. WDA is asymmetric and only monitors and protects the uplink toward the Web farm, which is the typical bottleneck in DDoS attacks. Legitimate traffic toward Web farms is very distinctive since it is produced by humans using Web-browsing software. Specifically, such upload traffic has low volume, and more importantly, has long off times that correspond to human view time. WDA utilizes these properties of legitimate client traffic to distinguish it from attack traffic, which tends to be continuous and heavy. A key feature of WDA is in its use of randomized thresholds that trap and penalize deterministic zombie traffic that tries to mimic human client patterns. WDA’s heart is WDAQ, a novel active queue management mechanism aimed to prefer legitimate client traffic over attacker traffic. With WDA installed, the attacker traffic toward the victim is attenuated. Extensive simulation results show that WDA can defeat simple flooding attacks, and can attenuate the bandwidth usable by sophisticated WDA-aware attacks by orders of magnitude. As a consequence, the attacker must increase his “bot-net” size by the same factor, to compensate for the effects of WDA. Our simulations show that WDA can defend a typical Web farm from DDoS attacks launched by hundreds of thousands zombies, while keeping legitimate clients’ service degradation under 10%.

1. Introduction

1.1. Background

Distributed Denial Of Service (DDoS) attacks are familiar threats to Internet users for more than 10 years. Since the year 2000, significant research efforts have been made in search of defense mechanisms. However, despite these efforts, enormous numbers of DDoS attacks are still taking place. According to security companies reports [2], in the second half of the year 2005, an average of more than 1400 attacks took place each day. This is a growth of more than 50% from the first half of the year.

In early DOS attacks only few a computers were involve in the attack, so the attacker typically spoofed the source IP address, to avoid being filtered. Since IP Spoofing was common in early DOS attacks, significant research was made in an attempt to find countermeasures and to track the attacker’s true address (cf. [46,44]).

In recent years, the “bot-net” phenomenon has came out to be a major contributor to unwanted and malicious Internet traffic [30]. This means that current distributed DOS attacks use thousands of infected zombie hosts that do not belong to the attacker and are very difficult to trace back to him. Hence, in current DDoS attacks, IP spoofing is a much smaller concern than it was—making previous anti-spoofing trace methods less relevant. Furthermore, since

* Corresponding author. Tel.: +972 3 640 6316.
E-mail addresses: ehudoron@eng.tau.ac.il (E. Doron), yash@acm.org (A. Wool).

1389-1286/$ - see front matter © 2011 Published by Elsevier B.V.
doi:10.1016/j.comnet.2010.05.001

they do not use spoofing, ‘‘bot-nets” can be used to mount more sophisticated attacks than simple flooding: The army of zombie hosts can open regular TCP sessions and issue legitimate-like HTTP requests, making a DDoS attack very difficult to distinguish from legitimate ‘‘flash crowds”. This is the backdrop for our system, which is designed to protect Web farms against modern DDoS attacks.

1.2. Web traffic and Web farms

The basis for WDA's protection is a characterization of upload Web traffic. Due to human behavior, upload Web traffic toward a Web farm is typically very bursty (‘‘ON–OFF nature”). After a Web user downloads a Web page and its embedded components to her Web browser, she reads the page or thinks about what she sees, then downloads another page, and so on. The time between such ‘‘Web sessions”, usually called the ‘‘viewing time”, can be estimated in seconds to tens of seconds. In addition, a legitimate Web session's upload traffic typically has low bandwidth: the traffic in the upload direction consists of TCP and HTTP session set-up, and TCP ACK packets, all of which are very short (see Section 2). In contrast, the traffic in the opposite (download) direction is typically high-bandwidth since it contains the actual data downloaded from the server. Our WDA system uses these observations to distinguish between legitimate and attack traffic.

A typical structure of Web farm is depicted in Fig. 1. The Web farm is connected to the Internet through a transport link which connect the Web farm to an ISP. This transport link is typically leased on a bit/s basis, with data rates ranging between hundreds of Mbit/s to a few Gbit/s. From the ISP, the traffic goes through routers, firewalls and load-balancers until it reaches the actual servers. It is quite reasonable to assume that the routers, internal switches and firewalls are all provisioned to have sufficient throughput to sustain flash crowds, and consequently, are less likely to be the bottleneck at the time of a DDoS attack. We argue that the likely bottleneck elements are either (a) the transport link from the ISP, or (b) the load balancer. Unlike internal networking cabling and equipment, the leased line to the ISP is a relatively costly part of the Web farm that is unlikely to be over-provisioned since it is a recurring expense. As for the load-balancer, many typical commercial products are engineered so the supported upload throughput toward the Web farm is two orders of magnitude lower than supported the throughput in the opposite direction. As a consequence, we argue that the load-balancer or ISP link tend to be flooded before all other components. Thus, in our architecture, we suggest placing the WDA as part of the ISP router's egress queue.

1.3. Contributions

This paper introduces WDA (*Web farm DDoS Attack attenuator*): a novel architecture to attenuate the DDoS attacker's bandwidth. WDA is especially designed to protect Web farms. WDA is asymmetric and only monitors and protects the uplink toward the Web farm, which is the typical bottleneck in DDoS attacks.

Defending the upload toward Web farm is achieved by WDA's main part: the *Web farm DDoS Attack attenuator Queuing* (WDAQ), a specialized active queue management mechanism that utilizes the characteristics of legitimate upload traffic to favor it over attack traffic. With WDA installed, the attacker traffic toward the victim is attenuated.

The first obvious challenge we face is defeat ‘‘regular” flooding attacks (including those that spoof IP addresses). Beyond these attacks, an important goal for WDA is to overcome DDoS attacks that employ ‘‘legitimate-like” traffic, issuing a high volume of properly formatted TCP and HTTP requests. We consider rather sophisticated attackers, which we assume to be fully aware of WDA's defense strategies, and are able to orchestrate a high degree of coordination over the zombies. To combat such sophisticated attackers, WDA uses randomized thresholds that trap and penalize deterministic zombie traffic that tries to mimic human client patterns.

WDA's main goal is to *attenuate* the DDoS attack bandwidth, and consequently to cause the attacker to increase his ‘‘bot-net” size as much as possible. Therefore, WDA main success criteria is the ratio between the ‘‘bot-net' size the attacker needs to assemble to overwhelm the Web farm when WDA is installed, versus the necessary ”bot-net” size without WDA.

Extensive simulation results show that WDA can defeat simple flooding attacks, and can attenuate the bandwidth usable by sophisticated WDA-aware attacks by orders of magnitude. As a consequence, the attacker must increase his ‘‘bot-net” size by the same factor, to compensate for the effects of WDA. Our simulations show that WDA can defend a typical Web farm from DDoS attacks launched by hundreds of thousands zombies, while keeping legitimate clients' service degradation under 10%. We made our simulation software freely available to the research community [9].

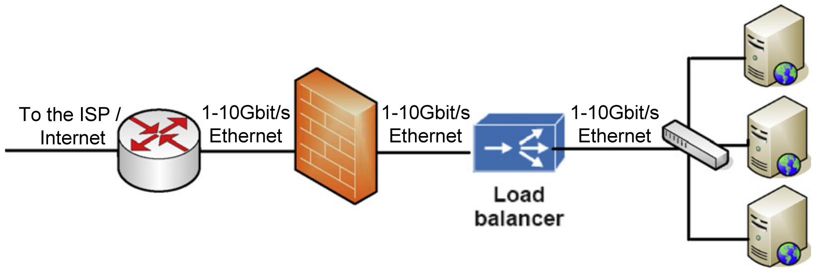


Fig. 1. Web farm basic structure.

Organization: The next section describes how we model Web traffic, and in particular its upload component. In Section 3 we present the design of WDA in detail. Our performance evaluation study is described is Sections 4 and 5. A survey of related works in the DDoS area is in Section 6. Caveats and subjects for future research are discussed in Section 7, and we conclude with Section 8. Additional details can be found in the Appendix.

2. Web traffic modeling

2.1. Basic Web traffic patterns

As we mentioned, WDA relies on legitimate Web traffic characteristics to favor it over attack traffic, thus attenuating the DDoS bandwidth. To do so we need an understanding of the patterns formed in Web traffic uplinks.

Web traffic patterns are created by a combinations of Web pages structure, Web browser implementation and user actions. A classic Web page consists of an HTML file with links to several other Web objects, that together compose a complete Web page. These Web objects can be categorized into two types: the "main object" and the "in-line objects". The HTML document is the main object (usually one per page), and the objects linked from it (style sheets, images, videos etc.) are the in-line objects (multiple per page).

Human interaction produces a fundamental property of Web traffic which is its "ON–OFF" nature. The HTTP OFF period represents the "thinking time" or "viewing time" of the human downloading the Web page. The HTTP ON period represents the Web page's total downloading time. The HTTP ON period ends after the last in-line object is completely downloaded. The HTTP ON and HTTP OFF together are called a "Web Session".

We now describe in more detail how HTTP works over a pipelined TCP connection (as in HTTP 1.1 [12]). The Web session starts with a TCP 3-way handshake opened by the client toward the server. The HTTP ON period begins with the client's Web browser sending an HTTP Get request for the page's main object. Through the open connection the client transmits the HTTP Get command, and receives a TCP ACK. The Web server generates an HTTP Response (on the download link) with the main object. The client's Web browser then processes the main object, and generates additional pipelined HTTP Get requests for each in-line object, through the same TCP connection (but see note 2 below). The server responds by transmitting the various in-line objects, while the client sends TCP ACKs for all the segments of the download traffic for all these objects.

Note that HTTP Responses typically require multiple IP packets, so the client transmits multiple TCP ACKs in the course of the in-line object reception. Furthermore, modern browsers transmit one TCP ACK after a successful reception of two IP packets [5].

Notes

(1) The above description is reasonable for "regular" Web content (e.g., on sites like CNN, Google etc.). Uplink traffic patterns for sites that use technologies like RSS and AJAX, or that allow video and audio upload (like YouTube) may not exhibit the same ON–OFF behavior. This is since the client's requests are not the immediate result of human actions, but of the automated script running inside the page. Therefore, WDA is not designed to protect such Web sites.

(2) Modern browsers such as Internet Explorer 6 and later in fact use multiple concurrent TCP connections to "Get" the in-line objects—typically 4 concurrent TCP connections. This implementation detail does not affect WDA, since, as we shall see, WDA keeps its counters per *client* (i.e., per source IP address), so it will correctly attribute the activities in all the concurrent connections to the same client's ON period.

(3) This model does not take into account clients that connect through a Web proxy or a NAT (Network Address Translation) box. In such cases the server will see multiple client connections originating from the same source IP address. This detail actually helps WDA: if the attacker places multiple zombies behind the same NAT box—WDA will view their aggregate traffic as coming from one aggressive client and attenuate all of them together.

2.2. Quantitative models

For the design of WDA, and for a realistic performance evaluation, we need a quantified statistical model of Web traffic. Models such as [3,7] are based on statistical analysis of extensive Web traffic traces. Since we care mostly about the upload parameters such as the viewing time and the amount of bandwidth from the Web clients to the Web server ("upload bandwidth"), we chose to adopt Choi and Limb's model [7] because it models not only the size and number of Web objects, but also the size of the HTTP Get messages, a critical parameter of the traffic from client to server. Table 4 in the Appendix shows the Choi–Limb model. For simplicity, only the parameters relevant for WDA are present.

Note that the Choi–Limb study [7], and other major studies of Web traffic such as "Surge" [3], are more than 10 years old already. We were unable to find quantified statistical models that are based on more modern traffic traces. Obviously, the Internet has evolved since these papers were written. In particular, the number of in-line objects they found seems exceedingly low: websiteoptimization.com [43] observed that the number of in-line objects is typically much larger than the mean of 5.55 reported in [7].

Therefore, we decided to use a modified model in which the number of in-line objects still follows a Gamma distribution but with a mean of 55.5 (i.e., an increase by factor of 10). As we shall see in the following section, increasing the average number of in-line objects forces WDA to allow a larger upload bandwidth for legitimate Web sessions—so it makes it harder for WDA to attenuate attacker traffic. We believe that using Choi and Limb's parameters "as-is" would have biased our simulation results too favorably.

Note that Web sites with high-upload bandwidth or with huge content for downloading (such as YouTube) most probably don't behave according to Choi–Limb model. As we noted before, WDA is not designed to function as DDoS attacks attenuator for these type of Web sites. See also the discussion in Section 7.

3. The design of WDA

3.1. Overview

WDAQ is an active queue managements mechanism designed to discriminate between legitimate Web clients' traffic and zombies' traffic. The WDAQ block diagram is depicted in Fig. 2. The policer is the main part of WDAQ. Packets forwarded to an egress queue of the ISP router enter the policer. The policer's task is to decide whether the incoming packets are "legitimate" or "suspicious". To do so, the policer maintains state on a per client basis, i.e., per source IP address. The policer sends "legitimate" packets to the high queue, and "suspicious" packets to the low queue. Both queues are common to all clients. The queues are scheduled using a standard Weighed Fair Queueing (WFQ) [29] policy, with weights that are strongly biased in favor of "legitimate" traffic. Note that "suspicious" traffic is *not* discarded automatically, as will be explained below. When traffic is light then the WFQ can serve both queues without loss. However, when traffic exceeds the link capacity, the WFQ causes packets in the low queue to be dropped.

Therefore, WDAQ's goal is to send only legitimate traffic to the high queue, and to send attacker traffic to the low queue. On the other hand, the attacker's goal is to overwhelm the high queue. Under WDAQ, this means that a sophisticated attacker should try to masquerade his traffic so WDAQ flags it as "legitimate". However, due to the sparse ON–OFF nature of legitimate Web upload traffic, forcing an attacker to send legitimate-like traffic makes each zombie much less aggressive, and attenuates the overall effective traffic the attacker can send.

Note that traffic sent to the low queue does not interfere with high-queue performance because of the way the WFQ scheduler is configured. So it is pointless for an attacker to try and flood the low queue.

3.2. The policer

As we saw in Section 2, a major characteristic of Web upload traffic from a specific client to a server is its "ON–OFF" nature. The WDAQ traffic policer makes the most of this property to limit the amount of traffic client sent to the high queue, and to the Web farm, during a single Web session. The policer monitors the total upload bandwidth, per Web session, and also the length of each clients' OFF periods, to support its decisions.

3.2.1. OFF period analysis

The first component of this bandwidth limitation is to enforce the OFF periods, on a per client basis. By "enforce" we mean to best serve clients that exhibit the expected OFF periods in their upload traffic streams, and to punish clients with a continuous transmission, i.e., to attenuate a continuous flooding from an attacker. As we saw in Choi and Limb's model, the thinking time is modeled with a Weibull distribution (recall Table 4 in the Appendix). This means that thinking time values vary over a large scale with significant probabilities: the thinking time median is a relatively short 11.7 s, but the standard deviation is 92.6 s. Therefore, we cannot impose hard thresholds on each individual OFF period length. It is easy to find scenarios of legitimate user actions that produce very short OFF periods—such as requesting a new page before the current page is downloaded, or rapidly clicking on links until getting the requested page, or just using the browser's "back" bottom. Nevertheless, WDAQ does use the *Average Off Period* (*AOP*) in its decisions. The *AOP* is calculated with a simple Exponential Weighted Moving Average between the current OFF time value and the average on all past sessions, as follows:

$$AOP[i+1] = (1 - W_{OP}) \cdot AOP[i] + W_{OP} \cdot OP[i], \qquad (1)$$

where:

- $AOP[i]$—The Average OFF Period (in seconds) for Web session number i. $AOP[i]$ is upper bounded with *AOP_High_TH*.
- $OP[i]$—The measured OFF Period (in seconds) in Web session number i.
- W_{OP}—A constant weight smaller than 1.

A client that exhibits a sequence of Web sessions with relatively short OFF times decreases his *AOP*. If the client's *AOP* drops below a predefined threshold, the client's next packets receive inferior service, i.e., they may be sent to the low queue. The predefined threshold is denoted by *AOP_Low_TH*. Because of the OFF Period enforcement the attacker must stop his transmission for a while, if he wishes to overload the high queue. As a consequence, the total

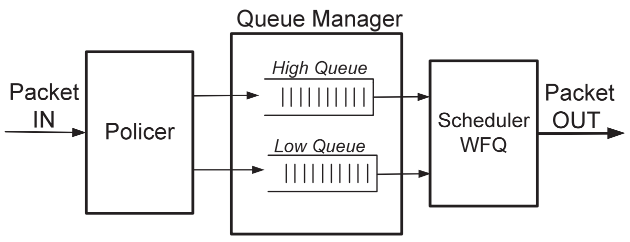


Fig. 2. WDAQ block diagram.

bandwidth the attacker can send to the high queue is diminished.

3.2.2. ON period analysis

The second policer task is to limit the amount of upload bandwidth sent to the high queue during Web page downloading.

The expected upload bandwidth in a legitimate ON period is composed of HTTP Request messages and the TCP ACK's needed while downloading the Web page and its components. Both are tied to the number of in-line Web objects. From Table 4 in the Appendix we see that the expected bandwidth is modeled by a heavy tailed distribution (LogNormal). The major challenge here is to find a bandwidth threshold that is large enough not to harm legitimate users, and conversely, not so large as to enable an attacker to inject too much traffic into the high queue. Our choice is to use a *dynamic* bandwidth limiter. The main idea here is to temporarily allow clients to enter more traffic than expected, but not constantly over time. The dynamic session bandwidth threshold depends on the client's behavior in past Web sessions. If a client sends more than the "expected" traffic to the Web farm in previous Web sessions, his bandwidth threshold for the current Web session is decreased. On the other hand, if he enters less bandwidth than expected, his bandwidth threshold for the current session is increased. The current session threshold is calculated as follows:

$$S_{TH}[i+1] = BF[i] \cdot S_{TH}[i], \tag{2}$$

where:

- $S_{TH}[i]$—The session bandwidth threshold (in bytes) for Web session number *i*. $S_{TH}[i]$ is upper bounded with *Session_Max_TH* and lower bounded with *Session_Min_TH*.
- $BF[i]$—The *behavior function* of the client in Web session number *i*.

The *behavior function* represents the client's historical behavior. A $BF[i]$ value larger than one mean a legitimate behavior, while values smaller mean suspicious behavior. The calculation of $BF[i]$ is described in the next section.

The actual bandwidth limitation is achieved as follows. In each Web session, the policer counts the traffic each client sends to the Web farm. All traffic up to $S_{TH}[i]$ bytes is sent to the high queue. Traffic that exceeds $S_{TH}[i]$ is sent to the low queue. The initial value of $S_{TH}[0]$ is set to *Session_Max_TH*, which has to be chosen such that it satisfies the upload bandwidth demand of a large Web page.

Note that it is possible for legitimate clients to occasionally exceed their session threshold. As long as there is no active DDoS attack this type of false-positive is not a real problem since traffic from the low queue does get served, so the client will be able to complete her download with some degradation of performance.

3.2.3. The dynamic bandwidth limiter: calculating the behavior function $BF[i]$

The calculation of the *behavior function* $BF[i]$ is based on the measured client traffic toward the Web farm $BW[i]$, but also on the current session threshold $S_{TH}[i]$, and on the historical average off period $AOP[i]$. As seen in Eq. (2), the next session thresholds $S_{TH}[i+1]$ is calculated from the current $S_{TH}[i]$, multiplied by a *behavior function* $BF[i]$. In general, well-behaved Web session receive a threshold increase ($BF[i] > 1$), while misbehaving sessions received a reduction ($BF[i] < 1$).

The precise calculation of $BF[i]$ is as follows. If the Web session bandwidth, $BW[i]$, exceeds the session threshold $S_{TH}[i]$, then $BF[i]$ is set to *suspicion_factor* (< 1). If the current Web session bandwidth is below the threshold, but the *AOP* is too short (below *AOP_Low_TH*) then $BF[i]$ is chosen uniformly at random between *ran_BF* and one. Finally, if the current bandwidth is below $S_{TH}[i]$ and $AOP[i]$ is long enough, this indicates a well-behaved session and $BF[i]$ is set to *legitimacy_factor* (> 1). The calculation is summarized in Eq. (3):

$$BF[i] = \begin{cases} \mathit{suspicion\_factor} & \text{if} \quad BW[i] > S_{TH}[i], \\ \sim U[\mathit{ran\_BF} \ldots 1] & \text{if} \quad BW[i] \leqslant S_{TH}[i], \\ & \text{and} \quad AOP[i] < \mathit{AOP\_Low\_TH}, \\ \mathit{legitimacy\_factor} & \text{if} \quad BW[i] \leqslant S_{TH}[i], \\ & \text{and} \quad AOP[i] \geqslant \mathit{AOPLowTh}. \end{cases} \tag{3}$$

Notes

(1) The session threshold, $S_{TH}[i]$, can grow only up to *Session_Max_TH*—to deny the attacker the option of remaining silent for a while to grow his bandwidth threshold exponentially.
(2) The *legitimacy_factor* and *suspicion_factor* are selected such that $1/\mathit{suspicion\_factor} > \mathit{legitimacy\_factor}$. This ensures that it is not advantageous for an attacker to cross his thresholds in one session, and hope to obey them in the next. If the attacker tries this approach his initial threshold will be restored one session (or more) later. As a consequence, to total bandwidth the attacker injects into the high queue is less than what he could have injected without crossing the threshold.
(3) Under certain conditions WDA keeps $S_{TH}[i]$ above the minimal threshold, *Session_Min_TH*. More on this below.

3.2.4. Defeating ON–OFF attackers with randomized trap sessions

The WDAQ mechanisms described so far suffice to defeat "simple flooding" attacks (that continuously transmit in full capacity): After crossing the initial bandwidth thresholds, all future attack packets are sent to the low queue, so concurrent legitimate traffic can flow freely. As a consequence, a smart attacker will prefer not to cross his initial threshold (*Session_Max_TH*). The OFF Period it also an obstacle for this attacker, because he has to stop his zombie transmission toward the victim, which reduces the attack intensity.

However, an intelligent attacker that controls his zombies to maliciously "live on the edge" is difficult to defeat using the method described so far. We call this adversary the “high-burst slow” attacker. The zombies of such an attacker exhibit an ON–OFF behavior: they periodically transmit *Session_Max_TH* bytes, and rest for *AOP_Low_TH* seconds. This pattern will be seen as legitimate behavior by the mechanisms we saw so far.

The key point that lets WDA combat this attack is that it is *deterministic*. Hence, we use randomization to our advantage, with the idea of *Trap sessions*. A *Trap session* works as follows: occasionally the policer randomly chooses a Web session in which it ignores the real measured values (bandwidth or OFF period) from the client. Instead, the policer selects random, and relatively “suspicious”, values, and uses them in the computation of the client’s *behavior function*. These artificial measurement values will cause WDAQ to momentarily reduce the client’s thresholds below *AOP_Low_TH* or *Session_Max_TH*. When the trap session is lucky and traps a “high-burst slow” zombie, that zombie’s threshold will quickly deteriorate to the minimum threshold and not recover, so it’s future traffic will be sent to the low queue. However, a legitimate client that is accidentally trapped will easily regain his session thresholds since his traffic exhibits legitimate behavior.

Specifically, WDA designates a session to be a trap session at random with probability of *Trap_Session_TH*. A trap session can be either a *BW* trap session or a *Think Time* trap session, with probability 1/2. In BW trap sessions WDA sets $BF[i]$ to a uniformly distributed random value in the range *BW_Trap_Session_Min* till *BW_Trap_Session_Max*. In Think Time trap sessions the current OFF-period value $OP[i]$ is chosen uniformly at random from the range *Think_Time_Trap_Min* till *Think_Time_Trap_Max*. These thresholds are constants that have to be chosen such that the trap session cause a minor degradation to legitimate client performance.

3.2.5. Minimum state

The WDAQ mechanisms we described so far suffer from a tendency to occasionally punish legitimate sessions in the following scenario: If a session happens to issue several HTTP requests in quick succession, or to exceed its session bandwidth threshold a few times in a row, WDAQ reduces the session threshold exponentially fast—and if left untreated, the threshold may drop so low that the session will keep exceeding it on all future requests. Therefore, we decided that the session bandwidth threshold must have a minimal lower bound of *Session_Min_TH* bytes.

Unfortunately, this bound could be abused by an adversary using the following strategy: repeatedly send *Session_Min_TH* bytes, then rest for a few seconds. We call this attacker a “low-burst fast” attacker. This attack strategy is not very aggressive since it transmits a relatively low bandwidth and stops between bursts, but without additional mechanisms the attack can persist indefinitely. Hence, we added dedicated means to handle this *minimum state* situation.

First, the session threshold $S_{TH}[i]$ is set to the lower bound *Session_Min_TH* only if the client has an *AOP* larger then *AOP_SUS_TH*. We ensure that $AOP\_SUS\_TH < AOP\_Low\_TH$, so it is unlikely for a legitimate client to exhibit such a low *AOP*. Second, a legitimate client would only need the protection offered by the lower bound in extreme cases and not constantly. On the other hand, the “low-burst fast” attacker would need the lower bound constantly. Therefore the policer allocates credits to each client that allow it to reach the minimum state (and have the lower bound protect its session threshold). The policer allocates *Minimum_State_Credits* credits to each client every 50 Web session. Therefore, in order to enter the minimum sate the client needs to meet these requirements: (a) $S_{TH}[i] < Session\_Min\_TH$, (b) $AOP > AOP\_SUS\_TH$, and (c) having a positive number of minimum state credits.

Once the client enters the minimum state, it obtains: $S_{TH}[i+1] := Session\_Min\_TH$ and $AOP := AOP\_Low\_TH$ (only if his $AOP < AOP\_Low\_TH$). One credit is taken from the client for each minimum state he entered. The number of accumulated credits is bounded by *Minimum_State_Max_Credits*, to prevent attackers from mounting a “low-burst fast” attack strategy even for a limited period. As a consequence of the minimum state defense mechanisms, once the “low-burst fast” attacker uses all his credits, he will fail to meet the minimum state requirements and his remaining traffic will be sent to the low queue.

4. Policer simulation and performance

4.1. Legitimate traffic in isolation

An important part of WDAQ are the values of the constants. In order to determine the constants, we ran a dedicated simulation only on the policer. The simulation was implemented using the NS2 [28]. We simulated the policer of a single legitimate client. The input to the simulation was a series of legitimate client’s upload traffic demands, and his thinking time for half a million Web sessions. The inputs were randomized across the distribution, according to the Choi–Limb model. These values were used as $BW[i]$ and $OP[i]$ in the policer equations. We measured the average amount of bytes sent to the low and to the high queues. Our main simulation objective was to find a set of constants that sends less than 1% of the legitimate to the low queue, i.e., a false-positive rate below 1%.

We paid special interest to the *Session_Max_TH* parameter, which controls the maximum byte count a client is allowed to upload (without punishment) during a Web session. This value is strongly related to the size of the Web pages (main + in-line) in the Web farm protected by WDA. To help calibrate this value, we measured the actual upload bandwidth required to download some popular Web sites (see Table 1). We measured the upload bandwidth using Internet Explorer (without any files in its cache). The inner pages were randomly chosen.

Table 1
Upload bandwidth required to view some popular Web sites.

Web site	Home page (Kbyte)	Inner page (Kbyte)
Amazon	81	71
Google	4.2	–
CNN	78	90
Yahoo!	24	50
eBay	38	40

Based on this examination we selected a relatively high value of 100 KB for *Session_Max_TH*—this is about four time larger than the mean upload bandwidth value which we calculated out of the Choi–Limb model (details omitted).

4.2. Quantifying the attack strategies

We also need to set the WDA constants to achieve effective attenuation against attackers. To this end, we simulated the behavior of the WDAQ policer, in isolation, against a single attacker following various attack strategies. For each strategy we calculated two metrics: the "Aggressiveness" and the "Attenuation".

The "Aggressiveness" metric measures the fraction of time that the attacker zombie is actually transmitting. Aggressiveness close to one means that the attacker is transmitting almost continuously. Specifically:

$$\text{Aggressiveness} = \frac{\text{ON time}}{\text{ON time} + \text{OFF time}}. \quad (4)$$

WDA imposes limitations on upload bandwidth (in bytes). Thus our various attack strategies are also defined in terms of upload bytes per burst. In order to calculate the Aggressiveness metric we need to convert these burst byte-counts into ON-period seconds. To do this we assume that the attack zombie has a 128 Kbps uplink bandwidth—the smallest uplink speed of current ADSL lines. Note that a zombie using the same strategy using a higher uplink speed will actually have a *lower* Aggressiveness score, which matches our intuition: the zombie can finish its upload burst faster, but its OFF period is still the same number of seconds, so on average it uses a smaller fraction of its available bandwidth.

The "Attenuation" metric measures the success of WDA against the attacker (in isolation). The intuition here is that attack traffic that reaches the low queue is harmless, so it is a waste of the attacker's time. Only the attack traffic that reaches the high queue is effective from an attacker's perspective. Thus we define the Effective-ON time as the total time (in seconds) that the attacker spent on transmitting traffic that reached the high queue. With this, we define Attenuation as:

$$\text{Attenuation} = \frac{\text{ON time} + \text{OFF time}}{\text{Effective-ON time}}. \quad (5)$$

4.3. Simulated attackers in isolation

We simulated WDA against the following attack strategies: (1) a "simple flooding" attacker, that transmits constantly; (2) the "high-burst slow" attacker described in Section 3.2.4. (3) the "low-burst fast" attacker described in Section 3.2.5; To these attack strategies we added two more low-aggressiveness strategies called the "low-burst slow" attacker and the "random" attacker. Note that these attack strategies are WDA-aware and were designed specifically to try to defeat the WDA mechanisms.

The intuition behind "low-burst slow" attacker is to avoid WDA's Trap sessions and minimum state handling mechanisms. Therefore the attacker periodically transmits *Session_Min_TH* bytes, and is quiet for *AOP_Low_TH* seconds. This attacker never enters the minimum state, and can easily recover from trap sessions because of his low upload rates.

The "random" attacker is also low-aggressive, and tries to use randomization in order to avoid the trap sessions. We assume that the attacker knows the values of the trap-session thresholds. The size of the "random" attacker's ON burst is chosen as follows: with probability *Trap_Session_TH* it transmits *Session_Min_TH* bytes, else it transmits a random-sized burst selected uniformly at random from the range $0.1 \times Session\_Max\_TH$ till $Session\_Max\_TH - 30KB$ bytes. The duration of the "random" attacker's OFF period is chosen as follows: with a probability of *Trap_Session_TH* he stops transmission for *AOP_Low_TH*, else he stops for the mean of the thinking time (39.5 s).

4.4. Policer parameters and results

In order to select the constant values we explored the parameter space as follows. We focused on the following pairs of parameters: (i) *legitimacy_factor* with *suspicion_factor*, (ii) *legitimacy_factor* with *ran_BF*, (iii) *Trap_Session_TH* with *Think_Time_Trap_Min*, (iv) *Trap_Session_TH* with *BW_Trap_Session_Min*, and (v) W_{op} with *AOP_Low_TH*. For each pair or parameters, we tested 5 settings per parameter (25 settings in total), while keeping all other parameters fixed. For each setting we ran a simulation of half a million Web sessions for each of the 6 isolated scenarios (one legitimate, 5 attacker strategies). As we mentioned before, our first criterion was to ensure a false-positive rate under 1% for legitimate traffic. For the attack traffic we tracked the Attenuation metric (with a goal of increasing it as much as possible).

As an illustration, Fig. 3a shows the dependence of the legitimate client's false-positive rate on *Trap_Session_TH* and *Think_Time_Trap_Min*. We can see that increasing the trap session rate (by reducing the think time or by increasing the threshold) increases the false-positive rate—however it remains under 1% for all settings except *Trap_Session_TH*=0.125. Conversely, Fig. 3b shows the dependence of the"random" attacker's attenuation on the same parameters. Here we can see that increasing the trap session rate *improves* the Attenuation. The parameter values we chose are compromises between these conflicting goals. The values for continued investigation for all parameters are summarized in Table 5 in the Appendix.

Table 2 shows that the "high-burst slow" attacker has an attenuation score of 233. Note that in the absence of the Trap session mechanism, this attacker strategy would only have been attenuated by factor of $1/\text{Aggressiveness} \simeq 4.2$. This demonstrates the effectiveness of the Trap session mechanism.

Furthermore, the "low-burst fast" attacker has an attenuation score of 101—instead of $\sim$12 without the minimum state handling mechanism.

Finally, Table 2 shows that the least aggressive attackers (the ones that transmit the least amount of traffic) are not affected very strongly by WDA: they have Attenuation $\simeq 1/\text{Aggressiveness}$, indicating that almost all their ON time is effective and most of their traffic reaches the high queue. But this means that WDA has already won

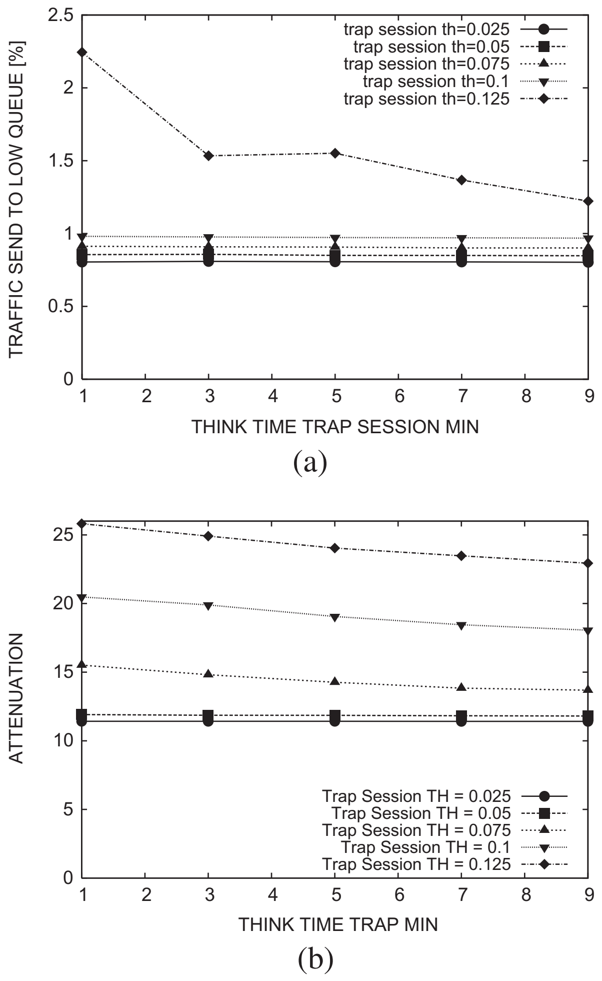


Fig. 3. (a) False positive rate (percent of legitimate traffic sent to low queue) as a function of *Think_Time_Trap_Min* for various values of *Trap_Session_TH*. (b) The "Random" attacker's attenuation, for the same parameters. In subsequent simulations we used the values *Trap_Session_TH* = 0.05 and *Think_Time_Trap_Min* = 5.

Table 2
Attackers characteristics and aggressiveness assuming a 128 Kbps upload, with the attenuation that WDA achieved.

Attacker name	ON period BW (bytes)	OFF period length (s)	Aggressiveness	Attenuation
Simple flooding	∞	0	1	∞
High-burst slow	100K	20	0.238	233
Low-burst fast	30K	10	0.158	101
Low-burst slow	30K	20	0.086	12.6
Random	[10K–70k]	[20–30]	0.083	11.7

the battle to a large extent: the attacker is forced to use very unaggressive strategies, so he would need an order-of-magnitude more zombies for an equivalent attack on a WDA-protected Web farm (and zombies with high-upload bandwidth are no better than those with basic ADSL lines).

5. Network simulation

Once we selected the parameter settings for WDA, and have an idea of the protection that it offers in isolation, we need to test WDA in a more realistic scenario. We want to simulate the effects of attack traffic (with various strategies) while legitimate traffic is being sent, for multiple legitimate clients and zombie armies, taking into effect the network topology, and the behavior of the TCP/IP stack. The simulation was also implemented with NS2 [28].

5.1. Simulation model

Fig. 5 (in the Appendix) shows the simulated Web farm structure. For practical reasons (so NS2 would be able to function) we simulated a scaled-down model. Instead of a typical 1 Gbps bottleneck link between the ISP router and the Web farm, we simulated a link of only 1.024 Mbps—i.e., scaled down by a factor of 1000. All our clients, both legitimate and attacker, were simulated with normal (unscaled) bandwidth rates typical of ADSL users: 1 Mbps download, 128 Kbps upload. Thus, in the scaled-down model, without WDA, 8 "simple flooding" attackers suffice to fill the bottleneck link to capacity and mount a successful DDoS attack: these 8 attackers represent a "bot-net" of 8000 zombies.

We simulated three legitimate clients (representing 3000), continuously downloading Web pages from the Web farm. The Web farm is simulated using a single NS2 node acting as the victim Web server. The "Web farm" suffers from a variety of DDoS attacks, generated by 0–200 simulated attackers (representing 200,000 zombies).

The Round Trip Time (RTT) for each client's path is modeled via the NS2 link's delay, and is chosen uniformly at random in the range 15–150 ms. Additional RTT delay is introduced by the "ISP" router queues. The choice of RTT values was based on the Internet Traffic Report [14].

Legitimate clients are modeled as NS2 nodes with TCP agents. The TCP version is the NS2 implementation for "TCP Vegas" with Delayed ACK [5]. We use the Delayed ACK because of its wide usage in today's Internet (and by Internet Explorer). On legitimate client nodes, we implement NS2 WEB traffic sources. The WEB traffic generator used in the simulation is based on Tom Henderson's contribution to NS2 [28]. We changed the original implementation of WEB traffic statistics to match Choi and Limb's model so the legitimate traffic is randomized across the distribution. Another change was made to support "pipelined HTTP 1.1" with a single TCP connection (the original implementation is for HTTP v1.0).

The attacker are also simulated as NS2 nodes, with Constant Bit Rate (CBR) traffic applications over UDP. Note that using UDP is just an NS2 convenience: we did not "filter" the attacks based on protocol. In other words, we simulated the attacker as a misbehaved TCP client: one that sends TCP packets on port 80 to the "Web farm", but does not obey TCP congestion avoidance mechanisms. All the attack packets have a length of 200 bytes.

WDAQ in implemented as the queuing discipline on the link from the "ISP" to the "Web farm". We implement WDAQ as a new C++ object in NS2. The WFQ implementation is based on Paolo Lossi WFQ contribution code. We used queue weights with a ratio of 4:1 (low queue = 0.2 and high queue = 0.8).

5.2. Basic simulation scenarios

In our simulation study we ran multiple scenarios to test WDA's ability to combat various attack strategies. Each scenario simulated 30,000 "simulation seconds" (just over 8 h). The main criterion measured by the simulation is the *number of successful Web sessions*—i.e., the number of successful page downloads that the legitimate clients accomplished. In all the scenarios the WDAQ constants were taken from Table 5.

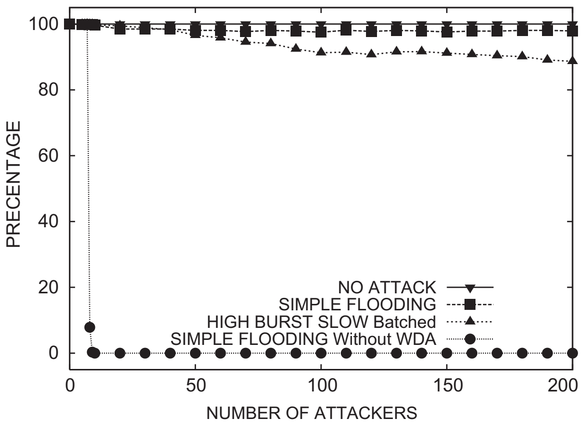
Scenarios number one and two are the "no attack" scenario: three legitimate clients without any attacker. In scenario one the queue in the transport link toward the Web farm is simple DropTail, while in scenario two we use WDAQ. In this scenario we measured an accomplishment of 2000 successful Web sessions (100%), see Fig. 4a. Scenario two gave identical identical results so it is omitted from the figure.

Scenario number three is "simple flooding" without WDA: we ran the same three legitimate clients, but this time they coexist with a "simple flooding" attacker with a growing "bot-net" size. The queue toward the Web farm is simple DropTail. The total attacker bandwidth grows with the number of attackers, so we except that 8–10 128 Kbps attacker will totally flood the DropTail queue of a 1.024 Mbps link. Fig. 4a indeed shows that about ten zombies totally deny service from the legitimate clients, and the fraction of successful Web sessions drops to 0% when the number of zombies grows any further.

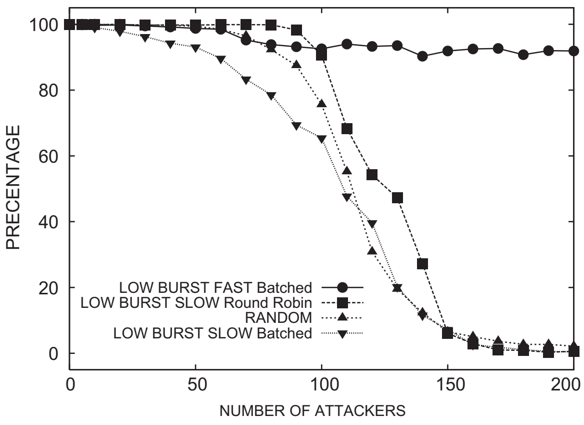
Scenario number four is another case of the "simple flooding" attacker but this time the queue toward the Web farm is WDAQ. Fig. 4a clearly shows that no matter how many attackers participate in the attack the legitimate clients hardly experience any degradation in their successful Web page downloads. Thus we see that WDA totally defeats the most common attack strategy available. If we scale-up the simulation results to a transport link with 1GBps, we can project that WDA can defeat an attacker with "bot-net" size of more than 200,000 zombies—and up to 2,000,000 zombies if the link runs at 10 Gbps.

5.3. ON–OFF attackers

In scenarios 5–8 we validated WDA against the intelligent ON–OFF attackers we already introduced: the "low-burst fast" and the "high-burst slow" attackers. For such attackers we need to consider how the attacker schedules his zombies during the OFF periods. We tested two scheduling variants for each attacker: a Batched variant and a Round-Robin variant. The variants differ when the number of zombies is too small to overwhelm the victim's upload bandwidth: In the Batched variant groups of zombies start



(a)



(b)

Fig. 4. (a) Percent of successful legitimate Web sessions as a function of the "bot-net" size for scenarios with no attack, and "simple flooding" attacks with and without WDA. (b) Percentage of successful legitimate Web sessions as a function of the "bot-net" size for scenarios with ON–OFF less aggressive attackers.

their ON pulse simultaneously to produce high spikes followed by quiet period, while the Round-Robin variant aims to produce a steady attack bandwidth. Note that both variants assume that the attack master has a very strict control over the zombies, and can coordinate their ON–OFF schedule.

Our results show that WDA performs very well against these both variants of these ON–OFF attackers. We observe that the fraction of successful legitimate Web sessions never drops below 85% even for 200 attackers (which represent a "bot-net" of 200,000 zombies). Details omitted due to space constraints.

5.4. Low aggressiveness ON–OFF attackers

Finally in the last three scenarios we validate WDA against the most difficult attackers to defend against (according to the analysis we saw in Section 4): the "low-burst slow" attacker and the "random" attacker. Scenarios number nine and ten deals with "low-burst slow" attacker. This attacker periodically transmits *Session_Min_TH* bytes (30 KB) and stops for *AOP_Low_TH* (20) seconds. As before, we check the two variants of attack coordination (Batched and Round-Robin). With the Low-burst Slow strategy 120 attackers are needed to achieve a "continuous" 1 Mbps transmission.

Because WDAQ hardly limits this attacker's bandwidth, we expect 120 zombies to totally flood the queue toward the victim. However, Fig. 4b shows that a total flooding occurs only with 150 zombies. The explanation here is related to the random RTT that the zombies introduce into WDAQ's high queue. Each time the ten active zombies halt their transmission and a new batch starts the queue experiences a burst of traffic. As a result, many of the attacker's packets are discarded from the queue, thus more legitimate client packets are able to enter to high queue successfully.

In the last scenario we examine the "random" attacker, as described in Section 4. Fig. 4b shows that in this scenario too the attacker needs over 150 zombies to completely block legitimate traffic.

5.5. Attack attenuation

When we try to quantify the "attenuation" in the network simulation we cannot use the simple metrics we used in the isolated simulation of Section 4.2. Instead we focus on a criterion which identifies situations where the service is actually denied. For this purpose we declare that the "service is denied" when legitimate clients have a Web-session success rate below 50%. With this definition the *Attack Attenuation* is the ratio between the "bot-net" size needed to "deny service" with WDA, and the "bot-net" size needed by a simple flooding attacker without WDA. Table 3 summarize these results.

Fig. 4a clearly show that for the "simple flooding", "high-burst slow", and "low-burst fast" attackers, the attack attenuation is infinite, because the number of successful Web sessions never drops below 50% regardless of the number of zombies.

Fig. 4b shows that about 110 zombies are needed to cause a "service is denied" condition when WDA is employed (there are minor differences between the "random" and the "low-burst slow" attackers). In comparison, when WDA is not employed, it suffices to use 7 simple-flood zombies to produce a similar "service is denied" condition. Thus, the attack attenuation we obtain against these attackers is 16–18.

Thus, we can see that the attack attenuation WDA achieves is 16–18 against very sophisticated attackers with high degree of zombie coordination, and WDA completely defeats simpler attackers, including ones that are aware of the defense mechanism and exhibit legitimate-like behavior. Note that zombies with higher upload speeds are not more useful to the attacker than those with normal upload speeds.

Table 3
Attackers aggressiveness and *attack attenuation* for 128 Kbps attackers. WDA defeats the aggressive attackers and attenuates the other by a factor of more than 16.

Attacker name	Aggressiveness	Attack attenuation
Simple flooding	∞	∞
High-burst slow	0.238	∞
Low-burst fast	0.158	∞
Low-burst slow	0.086	16
Random	0.083	18

5.6. Practical mechanism overhead

In order to evaluate the practicality of WDA we need to demonstrate that its data structures and algorithms impose a minimal overhead. This includes memory (RAM) overhead for the state, and processing overhead.

5.6.1. Memory

WDA maintains counters per sender—i.e., per source IP address. Furthermore, WDA does not inspect the packets and does not track the TCP state transitions. This means that WDA maintains much less information than firewalls or QoS filters do: such devices typically maintain state per connection—i.e., per tuple of (source IP, source port, destination IP, destination port). Specifically, for each source IP address, WDA maintains:

(1) The time (in ms) at which the last IP packet was received from this source IP.
(2) The number of bytes in the current ON period.
(3) The session threshold S_{TH}.
(4) The Average Off Period value (*AOP*).
(5) The *Minimum_State_Credits* counter.
(6) A status bit to indicate if the client has lost all its minimum state credits.

These counters can easily be kept in 18 bytes of RAM per source IP, possibly less. In other words, very modest RAM requirements.

5.6.2. CPU Overhead

As for CPU we separate the overhead into activities that must be completed at wire-speed, and must be implemented in hardware, versus activities that are less frequent and can be implemented in software.

The line-speed activities comprise of two mechanisms:

(1) If the time between two consecutive IP packets is more than 0.8 s, and the last packet was not dropped, then a new ON period is declared: send the time and byte counters to the software level for processing. If the client has not lost its minimum state credits, set the S_{TH} to *Session_Min_TH*, else S_{TH} is unchanged.
(2) Compare the number of bytes in the ON period to S_{TH}. If the number of bytes in the ON period is less than S_{TH} then send the packet to the high queue, else send it to the low queue.

These simple activities clearly consume a very small amount of hardware resources, which can easily be supported by a network processor of FPGA.

All the other WDA activities are not per-packet operations so they do not need to work at line speed. Specifically, when the hardware indicates that a new ON period has started, the software needs to read the hardware counters and implement the calculations outlined in Section 3. The CPU is required to accomplish this calculation in the time it takes an ADSL client to transmit *Session_Min_TH* bytes—a relatively long time (fractions of seconds). Thus we estimate that a current med-range CPU can handle a few million Web clients.

6. Related work

The area of DDoS bandwidth attacks defense schemes has been studied very intensively. Many defense strategies and directions have been introduced—see [27,10]. The following review attempts to classify earlier research.

The first class is a defense mechanism that analyzes the incoming traffic to a router and tries to mitigate or even to eliminate the attacks. A very comprehensive research by Mahajan et al. [23] introduce the Aggregate-based congestion control (ACC). In ACC the router dynamically classifies the incoming traffic into *aggregates*. The ACC controls those aggregates by limiting the amount of traffic they can insert to the network. The router can also Pushback [15] those limitations to its upstream routers (towards the sources of those packets), by ordering them to perform the limiting on the suspicious aggregates. The main concern with ACC is its inability to distinguish legitimate traffic from attack traffic. The rate limiting easily causes collateral damage.

Mirkovic et al. proposed D-Ward [26], the main idea here is to identify and suppress the attack at the source network. According to traffic type (TCP, UDP etc.), D-Ward finds flows with abnormality, and performs a rate limiting on them. This approach suffers from a number of disadvantages: it relies on the symmetry of the traffic and the bi-directionality of protocols; D-Ward can solve or mitigate DDoS only if all the networks in the Internet will deploy it; next, an ISP has no motivation to invest in implementing facilities that will not benefit his clients. Even when deployed, an DDoS attack with high number of well behaved zombies, will not be disturbed by D-Ward. A similar approach of dynamically measuring the ratio of ingoing and outgoing traffic is presented by Gil and Poletter in MULTOPS [13].

Wang and Shin [41] propose to treat DDoS as a traffic management problem. They propose to use a separate queue for each traffic type (TCP, UDP and ICMP) and, like in DiffServ, for each service level. In this way, attackers which flood the victim with UDP traffic, will do no harm to the TCP traffic. This mechanism is ineffective against an adversary that simply varies its traffic characteristics during the attack.

Thomas et al. in NetBouncer [38] propose a set of legitimacy tests in order to expose attackers. NetBouncer is deployed in the ingress of a network, close to the victims. NetBouncer challenges the sources of the incoming traffic in the IP layer (finds if a source exists), in the transport (TCP) layer (finds if a source IP is not spoofed) and in the application service layer (finds if a client is human or a zombie machine). The incoming traffic is served according to its sources' legitimacy. The main shortcoming of NetBouncer is that it needs to analyze and intervene in every possible protocol—and a smart attacker can plan an attack that can't be revealed by the tests. Nonetheless, we use some of the ideas of NetBouncer (and in particular the SYN-cookie) in WDA. A similar approach was introduced in [24], with a dynamic middlebox that performs bidirectional traffic interception to recognize and mitigate attacks.

In [6], Bremler-Barr et al. propose a new WFQ-like active queue management mechanism called Aggressiveness Protective Fair Queuing, that dynamically decreasing the queue weight of the aggressive users. The actual weight used for a flow is a dynamically varying variable reflecting the past bandwidth usage of the flow. This traffic control mechanism needs to be flexible enough to adjust to the time-varying demands of the polite bursty applications (like WEB traffic) while handling the aggressive applications (DDoS).

The second class is defense techniques that make use of *capabilities*. Yang et al. [47] and Yaar et al. [45] propose a novel network architecture in which the destinations of traffic have the ability to govern the traffic they receive. This approach suffers from deployment problems: The need to update all the hosts and routers in the Internet, makes the deployment of capability impractical. Furthermore, it relies on stable path between source and destination, which is not always maintained, specially during an attack.

The third class is defense techniques that make use of *puzzles*. The traffic sources are asked by the destinations, or routers toward them, to solve puzzles before they are authorized to send IP traffic to them. In [42], Wang and Reiter introduce a scheme of using puzzles by routers: When a router recognizes a congestion it asks the host which is sending data to continuously solve a computational puzzle. In [17], graphical puzzles are used to protect Web server against numerous number of legitimate liked service requests. The server asks the clients users to solve a graphical puzzle before it allocates any resources for those users. The graphical puzzle can be easily solved by human being and hard to be solved by a machine.

The fourth class uses the assumption that the statistics of traffic on a specific link is stable during regular operation and changes during DDoS attacks. The main idea here is to identify the changes in traffic characteristics and use them to differentiate attack packets from legitimate ones. In [19] Kim et al. use this approach to give each coming packet a score which is a measure of its reliability. The weakness of this approach is the ability of an intelligent adversary to overcome the proposed filters by varying its traffic characteristics over time [21].

LADS [32] is a large-scale automated DDoS detection system to indicate the presence of flow anomalies. LADS is based on a triggered multi-stage architecture for scalable attack detection. Conceptually, the initial stages consist of low-cost anomaly detection mechanisms that provide information to traffic collectors and analyzers to reduce the search space for further traffic analysis. Reval [39] assists network operators to evaluate the impact of DDoS attacks and identify feasible mitigation strategies in real-time, by modeling resource constraints of network elements and incorporating routing information.

The fifth class is a very large group of researches dealing with recognition of IP spoofing such as [44,46,4,11]. The main problem here is the ability of a "bot-net" owner to launch a devastating attack with a large enough number of zombies flooding the victim, without resorting to IP spoofing.

The last class uses overlay networks [18,35] to defend against DDoS attacks. Only predefined and authorized users are able to enter and use the overlay network. This solution is only suitable in specific scenarios where all the clients and servers are known in advance—an a-typical situation for Web farms. Additional works in this direction include [40,37,36].

Companies including Cisco [8], Arbor Networks [1], and Mazu Networks [25], have developed anti-DDoS attacks products. These products claim to use a mix of anomaly recognition, protocol analysis or smart rate limits, to remove malicious traffic while allowing good packets to pass. Unfortunately the designs and algorithmics of these products are not publicly disclosed so we cannot discuss them meaningfully.

Beside of defense techniques, some researchers introduced sophisticated DDoS attack methods. In [20], a low-rate DOS attack (*"shrew attack"*) is presented. The attacker maliciously sends a high volume "pulse" of traffic in a short period. The legitimate packets can't reach the victim, so it does not send ACK on them. This causes the sender to time out. In [22], the authors suggest to use the periodic nature of the attack to expose it using Wavelet transforms. A different approach presented by Sherwood et al. in [33]. The attacker here is the TCP receiver (client). The main idea is that the attacker sends ACK without waiting for the data packet, neutralizing the TCP congestion control, and causing the servers to transmit a high volume of traffic.

Another type of DDoS attacks targets server resources (such as disk and database bandwidth, memory, and CPU). In [34], server-side middleware is proposed to counter application level DOS attack. Ranjan et al. in [31] also assigned a continuous "suspicion" level to each incoming client. A DDoS-resilient scheduler utilizes these values to decide how to schedule a client's next session request. In [16] Jung et al. tried to distinguish DDoS attack from a legitimate flash crowd.

7. Caveats and future research

WDA is implemented very close to the attack's victims. Therefore, WDA cannot defend against attacks in which links become congested before reaching the Web farm. However, the advantage of the WDA architecture is that it gives a direct benefit to the ISP that operate WDA—since WDA provides value to the ISP's own paying customers.

Because WDA relies on the characteristics of human Web-browsing, it is not suitable for the protection of Web sites with high-bandwidth uploads of images or videos (like YouTube). Furthermore, Web sites using RSS, AJAX or SaaS do not have human Web-browsing attributes. Therefore, such Web farms require other protection mechanisms—which we leave for future research.

We note that having *Session_Max_TH* as a predefined constant is somewhat problematic. It would have been better to use a dynamic threshold that can calibrate itself to the size of Web pages in the protected Web farm. However, care must be taken that the mechanism for selection of the dynamic threshold is not a target of attack. This is also left for future research.

Beyond a simulation-based performance evaluation, it would be interesting to evaluate WDA's performance against traces taken from real Web farms which host popular Web sites. In fact, it would be valuable first to revisit the traffic model of Choi–Limb [7] and the 'Surge" [3] and to calibrate them to current-day Web traffic characteristics. To enable this future work we made our simulation software freely available to the research community [9].

We argue that since WDA does not perform conventional rate-based traffic shaping, it will not harm the performance of the Web farm in legitimate "flash crowd" scenarios. If a large number of human clients connect simultaneously, then their browsing characteristics will be those of humans—and we predict that WDA will keep sending all this traffic to the high queue. Note, though, that WDA will not reduce the load on the servers in any way, so it cannot be viewed as a "surge protector" against high volumes of legitimate traffic. It may be interesting to verify this argument in simulations.

WDA can easily be extended to handle "Shrew" attacks [20]. In such an attack the attacker exploits the retransmission mechanism of TCP to flood the attacked queue with short outage transmitted in a one second period. Each time the TCP stack of a client finishes its RTO and tries to retransmit (in one second granularity), it encounters the attacker-generated outage. As a result the legitimate client re-enters retransmission and suffers from long RTO periods. We observe that if the client's first packet after the RTO successfully evades the short outage, the following packets will receive a good service. Therefore, in order to defend against Shrew attacks, we suggest adding another queue to WDAQ: a super queue with highest priority. Only the "first" packet after a period of more than one second without transmission, is allowed to enter this queue. We believe that overwhelming this queue is quite difficult, so legitimate users will elegantly elude this pulsing attack. This idea needs to be validated.

Finally, in a cases of organizations deploying a NAT for better usage of IP address, the traffic from a specific IP address may not have the ON–OFF nature typical of a single human's behavior. This may happen especially in cases where many users from the same organization concurrently try to download Web pages from the same Web farm. Further research is needed in order distinguish the actual users without being exploited by an attacker.

8. Discussion and conclusions

This paper introduced WDA, a novel architecture to attenuate the DDoS attacker's bandwidth when attacking a Web farm. WDA monitors and protects the uplink toward the Web farm, which is the typical bottleneck in DDoS attacks. WDA relies on the fact that legitimate upload traffic toward Web farms is produced by humans using Web-browsing software. WDA uses the characteristics of such traffic (low bandwidth and long OFF periods) to punish zombie traffic, that tends to be continuous and heavy.

Beyond the simple flooding attackers that are currently prevalent, we also considered sophisticated attack strategies that try to exhibit legitimate-like behavior, and rely on an intimate familiarity with WDA mechanisms. To combat such strategies WDA uses randomized thresholds that trap and penalize *deterministic* zombie traffic that tries to mimic human client patterns.

Extensive simulation results show that WDA can defeat simple flooding attacks, and can attenuate the bandwidth usable by sophisticated WDA-aware attacks by orders of magnitude. As a consequence, the attacker must increase his "bot-net" size by the same factor, to compensate for the effects of WDA. It is known that the cost of renting a "bot-net", or for creating a private one, has significantly decreased in recent years—but not down to zero, e.g., it is estimated that renting an army of 20,000 computers costs approximately $2000. If WDA attenuation forces the attacker to rent 10-times more zombies to achieve the same DoS condition, his cost climbs to $10,000–20,000 (even assuming volume discounts)—which may well change the economic viability of the attack. Based on our simulations we project that WDA can defend a typical Web farm from DDoS attacks launched by hundreds of thousands zombies, while keeping legitimate clients' service degradation under 10%.

Appendix A. ON and OFF period detection

For WDAQ to function well the policer needs to decide, on a per client basis, when a new Web session starts and ends. This could be done by "deep packet inspection" methods that look into the HTTP payload and follow the protocol. However, these methods are relatively complex, and typically require following both the uplink and the downlink traffic. Instead, we propose a much simpler technique, whose accuracy is sufficient for the needs of WDAQ: Following "Surge" [3], when the policer sees a packet from a client that did not transit any packets for at least 0.8 s, the policer declares the just-ending quiet time an OFF period, and starts a new ON period. Otherwise, if the previous packet was seen less than 0.8 s ago—the current packet is counted as part of the current ON period.

This very simple mechanism does have some side-effects. First it causes WDAQ to never identify an OFF period that is *shorter* than 0.8 s. However, according to Choi and Limb's Web traffic model, such short OFF periods do occasionally occur. When this happens, WDAQ considers what should be two separate Web sessions to be a single session—potentially causing the merged session to exceed its bandwidth threshold. However, WDAQ is designed to handle infrequent situations in which S_{TH} is exceeded by legitimate sessions, so this scenario is not a major concern.

A more subtle scenario,which we discovered during the simulations, occurs during traffic congestion conditions that are typical of bandwidth attacks. Under congestion, legitimate packets may be discarded, causing TCP to retransmit them after the Retransmit Time Out (RTO). By default the RTO starts at 1 s and grows with exponential back-off. This causes two problems to WDAQ: first, the retransmitted packet is seen after a quiet time of at least 1 s (i.e., >0.8 s), so WDAQ marks an OFF period and starts a new ON period, thus splitting the session. But worse, this artificial RTO-produced OFF period is much shorter than *AOP_Low_TH* (recall that the mean OFF period in the Choi–Limb model is 39.5 s; we used $AOP\_Low\_TH = 20$ in our simulations). So the client's *AOP* is very likely to drop below the *AOP_Low_TH* causing more of its traffic to reach the low queue, where it is likely to be dropped again due to the same congestion we started with, exacerbating the situation.

In order to overcome this condition, we added the following refinement to the decision when a new ON period is declared: when a new packet arrives after a quiet period of over 0.8 s, if the client's *previous* packet was dropped from one of WDAQ's queues then a new ON Period is *not* declared.

Appendix B. Anti spoofing

Recall that WDA maintains state for each client—and clients are identified by their source IP address. Therefore, an attacker that executes a SYN-flood attack while spoofing its source IP address may eventually overwhelm WDA's state space. Thus an *Anti-Spoofing Mechanism* (ASM) is essential. For this purpose WDA employs the "SYN cookie" mechanism (cf. [38]). Every first packet sent by a new client is caught by the ASM. If this packet is a TCP SYN, the ASM answers to sender instead of the Web server (the real destination of the packet), with a TCP SYN + ACK packet. The TCP sequence number sent by ASM is a result of a cryptographic hash function with a secret key. If the sender's source IP address is not spoofed, the sender receives the SYN + ACK, and answers with an ACK packet that includes the ASM's sequence number as the ACK number. The ASM re-checks the cryptographic hash, so it can authenticate that the sender IP address is not spoofed. The ASM then open another TCP connection with the Web server, and interconnects the two connections. The ASM also manages a white-list of all the safe IP address it already encountered, so the above procedure is made only in the first connection. IP addresses are removed from the white-list using aging.

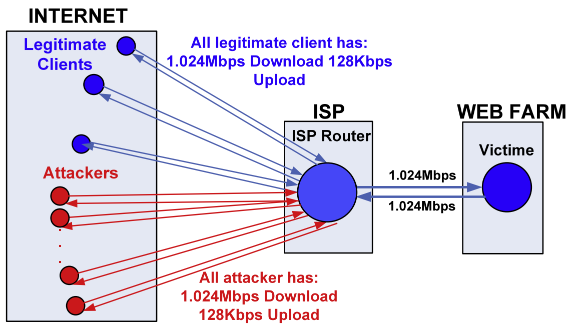

INTERNET
Legitimate Clients
All legitimate client has: 1.024Mbps Download 128Kbps Upload
ISP
ISP Router
WEB FARM
Victime
Attackers
1.024Mbps
1.024Mbps
All attacker has: 1.024Mbps Download 128Kbps Upload

Fig. 5. Simulation model.

Table 4
Choi–Limb statistics for Web traffic.

Parameter	Mean	Median	S.D	Distribution
Request size (bytes)	360.4	344	106.5	LogNormal
Main object size (bytes)	10,710	6094	25,032	LogNormal
In-line object size (bytes)	7758	1931	126,168	LogNormal
# In-line objects	5.55	2	11.4	Gamma
Viewing time (s)	39.5	11.7	92.6	Weibull

Table 5
Policer selected parameters.

Parameter	Value	Parameter	Value
legitimacy_factor	19/16	*suspicion_factor*	11/16
ran_BF	0.75	W_{op}	0.1
Session_Max_TH	100 KB	*Session_Min_TH*	30 KB
AOP_High_TH	90 s	*AOP_Low_TH*	20 s
Trap_Session_TH	0.05	*AOP_SUS_TH*	10 s
BW_Trap_Session_Min	0.4	*BW_Trap_Session_Max*	1
Think_Time_Trap_Min	5 s	*Think_Time_Trap_Max*	15 s
Minimum_State_Credits	4	*Minimum_State_Max_Credits*	200

References

[1] Arbor Networks: Arbor Peakflow SP – Data Sheet, <http://www.arbornetworks.com/peakflowsp>.
[2] Arbor Networks: Worldwide ISP Security Report, September 2005, <http://www.arbor.net/downloads/Arbor_Worldwide_ISP_Security_Report.pdf>.
[3] P. Barford, M. Crovella, Generating representative Web workloads for network and server performance evaluation, in: SIGMETRICS, 1998, pp. 151–160.
[4] S.M. Bellovin, ICMP Traceback Messages, Work in Progress, Internet Draft draftbellovin-itrace-00.txt, March 2000.
[5] R. Braden, IETF RFC 1122: Requirements for Internet Hosts – Communication Layers, October 1989.
[6] A. Bremler-Barr, N. Halachmi, H. Levi, Aggressiveness Protective Fair Queueing for Bursty Applications, in: IWQoS, 2006.
[7] H.-K. Choi, J.O. Limb, A behavioral model of Web traffic, in: ICNP, 1999, pp. 327–334.
[8] Cisco Systems: Defeating DDOS Attacks – White Paper, <http://www.cisco.com/en/US/prod/collateral/vpndevc/ps5879/ps6264/ps5888/prod-white-paper0900aecd8011e927.pdf>.
[9] E. Doron, WDA Source Code, 2010, <http://www.eng.tau.ac.il/~yash/wdaq-2010-04.zip>.
[10] C. Douligeris, A. Mitrokotsa, DDoS attacks and defense mechanisms: classification and state-of-the-art, Computer Networks 44 (5) (2004) 643–666.
[11] P. Ferguson, D. Senie, IETF RFC 2267: Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing, January 1998.
[12] R. Fielding, J. Getty, J. Mogul, H. Frystyk, T. Berners-Lee, IETF RFC 2616: Hypertext Transfer Protocol – http/1.1., June 1999.
[13] T.M. Gil, M. Poletter, MULTOPS: a data-structure for bandwidth attack detection, in: Proceedings of USENIX Security Symposium 2001, Washington, DC, August 2001.
[14] The Internet Traffic Report: Networks Overview, March 2008, <http://www.internettrafficreport.com/>.
[15] J. Ioannidis, S.M. Bellovin, Implementing pushback: router-based defense against DDOS attacks, in: NDSS, 2002.
[16] J. Jung, B. Krishnamurthy, M. Rabinovich, Flash crowds and denial of service attacks: characterization and implications for CDNs and Web sites, in: WWW, 2002, pp. 293–304.
[17] S. Kandula, D. Katabi, M. Jacob, A. Berger, Botz-4-sale: surviving organized DDOS attacks that mimic flash crowds, in: NSDI, 2005.
[18] A.D. Keromytis, V. Misra, D. Rubenstein, SOS: an architecture for mitigating DDOS attacks, IEEE Journal on Selected Areas in Communications 22 (1) (2004) 176–188.
[19] Y. Kim, W.-C. Lau, M.C. Chuah, H.J. Chao, Packetscore: statistical-based overload control against distributed denial-of-service attacks, in: INFOCOM, 2004.
[20] A. Kuzmanovic, E.W. Knightly, Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants, in: SIGCOMM, 2003, pp. 75–86.
[21] Q. Li, E.-C. Chang, M.C. Chan, On the effectiveness of DDOS attacks on statistical filtering, in: INFOCOM, 2005, pp. 1373–1383.
[22] X. Luo, R.K.C. Chang. On a new class of pulsing denial-of-service attacks and the defense, in: NDSS, 2005.
[23] R. Mahajan, S.M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, S. Shenker, Controlling high bandwidth aggregates in the network, Computer Communication Review 32 (3) (2002) 62–73.
[24] A. Mahimkar, J. Dange, V. Shmatikov, H. Vin, Y. Zhang, dFence: transparent network-based denial of service mitigation, in: Fourth USENIX NSDI, Cambridge, MA, April 2007.
[25] Mazu Networks: Mazu Enforcer – Product Sheet, <http://www.mazunetworks.com/resources/product-sheets/Mazu-ProductSheet-Enforcer.pdf>.
[26] J. Mirkovic, G. Prier, P.L. Reiher, Attacking DDOS at the Source, in: ICNP, 2002, pp. 312–321.
[27] J. Mirkovic, P.L. Reiher, A taxonomy of DDoS attack and DDoS defense mechanisms, Computer Communication Review 34 (2) (2004) 39–53.
[28] NS2, The Network Simulator, <http://nsnam.isi.edu/nsnam/index.php/User_Information>.
[29] A.K. Parekh, R.G. Gallager, A generalized processor sharing approach to flow control in integrated services networks: the single-node case, IEEE/ACM Transactions on Networking 1 (3) (1993) 344–357.
[30] M.A. Rajab, J. Zarfoss, F. Monrose, A. Terzis, A multifaceted approach to understanding the botnet phenomenon, in: Internet Measurement Conference, 2006, pp. 41–52.
[31] S. Ranjan, R. Swaminathan, M. Uysal, E.W. Knightly, DDoS-resilient scheduling to counter application layer attacks under imperfect detection, in: INFOCOM, 2006.
[32] V. Sekar, N. Duffield, O. Spatscheck, J. van der Merwe, H. Zhang, LADS: Large-scale Automated DDoS Detection System.
[33] R. Sherwood, B. Bhattacharjee, R. Braud, Misbehaving TCP receivers can cause Internet-wide congestion collapse, in: ACM Conference on Computer and Communications Security, 2005, pp. 383–392.
[34] M. Srivatsa, A. Iyengar, J. Yin, L. Liu, A middleware system for protecting against application level denial of service attacks, in: Middleware, 2006, pp. 260–280.
[35] A. Stavrou, D.L. Cook, W.G. Morein, A.D. Keromytis, V. Misra, D. Rubenstein, WebSOS: an overlay-based system for protecting Web servers from denial of service attacks, Computer Networks 48 (5) (2005) 781–807.
[36] A. Stavrou, J. Ioannidis, A.D. Keromytis, V. Misra, D. Rubenstein, A pay-per-use DoS protection mechanism for the Web, in: ACNS, 2004, pp. 120–134.
[37] A. Stavrou, A.D. Keromytis, J. Nieh, V. Misra, D. Rubenstein, MOVE: an end-to-end solution to network denial of service, in: NDSS, 2005.
[38] R. Thomas, B. Mark, T. Johnson, J. Croall, NetBouncer: client-legitimacy-based high-performance DDOS filtering, in: DISCEX, 2003, pp. 14–25.
[39] R. Vasudevan, Z. Mao, O. Spatscheck, J. van der Merwe, Reval: a tool for real-time evaluation of DDoS mitigation strategies, in: USENIX Technical Conference, 2006.
[40] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, S. Shenker, DDoS defense by offense, in: ACM SIGCOMM 2006, Pisa, Italy, September 2006.
[41] H. Wang, K.G. Shin, Transport-aware IP routers: a built-in protection mechanism to counter DDOS attacks, IEEE Transactions on Parallel and Distributed Systems 14 (9) (2003) 873–884.
[42] X. Wang, M.K. Reiter, Mitigating bandwidth-exhaustion attacks using congestion puzzles, in: ACM Conference on Computer and Communications Security, 2004, pp. 257–267.
[43] websiteoptimization.com, Average Web Page Size Triples Since 2003, <http://www.websiteoptimization.com/speed/tweak/average-web-page/>.
[44] A. Yaar, A. Perrig, D.X. Song, PI: a path identification mechanism to defend against DDOS attack, in: IEEE Symposium on Security and Privacy, 2003, p. 93.
[45] A. Yaar, A. Perrig, D.X. Song, SIFF: A stateless Internet flow filter to mitigate DDOS flooding attacks, in: IEEE Symposium on Security and Privacy, 2004, p. 130.
[46] A. Yaar, A. Perrig, D.X. Song, FIT: fast Internet traceback, in: INFOCOM, 2005, pp. 1395–1406.

[47] X. Yang, D. Wetherall, T.E. Anderson, A DOS-limiting network architecture, in: SIGCOMM, 2005, pp. 241–252.

Avishai Wool received a B.Sc. (Cum Laude) in Mathematics and Computer Science from Tel Aviv University, Israel, in 1989. He received an M.Sc. and Ph.D. in Computer Science from the Weizmann Institute of Science, Israel, in 1993 and 1997, respectively. He then spent 4 years as a Member of Technical Staff at Bell Laboratories, Murray Hill, NJ, USA. In 2000 he co-founded Lumeta corporation, a startup company specializing in network security, and its successor, Algorithmic Security (AlgoSec). He is currently an Associate Professor at the School of Electrical Engineering, Tel Aviv University, Israel, where he has been since 2002. He is the creator of the AlgoSec Firewall Analyzer. He has served on the program committees of the leading IEEE and ACM conferences on computer and network security. He is a senior member of IEEE, and a member the ACM and USENIX. His research interests include firewall technology, secure storage, computer, network and wireless security, smartcard-based systems, and the structure of the Internet topology.