DDoS-Resilient Scheduling to Counter Application Layer Attacks under Imperfect Detection

S. Ranjan, R. Swaminathan, M. Uysal, and E. Knightly

Abstract— **Countering Distributed Denial of Service (DDoS) attacks is becoming ever more challenging with the vast resources and techniques increasingly available to attackers. In this paper, we consider sophisticated attacks that are protocol-compliant, non-intrusive, and utilize legitimate application-layer requests to overwhelm system resources. We characterize application-layer resource attacks as either request flooding, asymmetric, or repeated one-shot, on the basis of the application workload parameters that they exploit. To protect servers from these attacks, we propose a counter-mechanism that consists of a suspicion assignment mechanism and a DDoS-resilient scheduler, *DDoS Shield*. In contrast to prior work, our suspicion mechanism assigns a continuous valued vs. binary measure to each client session, and the scheduler utilizes these values to determine if and when to schedule a session's requests. Using testbed experiments on a web application, we demonstrate the potency of these resource attacks and evaluate the efficacy of our counter-mechanism. For instance, we effect an asymmetric attack which overwhelms the server resources, increasing the response time of legitimate clients from 0.1 seconds to 10 seconds. Under the same attack scenario, DDoS Shield limits the effects of false-negatives and false-positives and improves the victims' performance to 0.8 seconds.**

I. Introduction

Distributed Denial of Service (DDoS) attacks pose an ever greater challenge to the Internet with increasing resources at the hands of the attackers. Recent studies estimate that farms of compromised hosts, popularly known as "botnets," are as large as 60,000 machines [9][21]. Moreover, the SYN flood attack, the most popular DDoS attack to date, is giving way to sophisticated application-layer (layer-7) attacks. In one instance, an online merchant employed the "DDoS mafia" to launch an HTTP flood towards his competitors' web sites by downloading large image files when a regular SYN flood failed to bring the site down [11].

Many prior attacks targeted network bandwidth around Internet subsystems such as routers, Domain Name Servers, or web clusters. However, with increasing computational complexity in Internet applications as well as larger network bandwidths in the systems hosting these applications, server resources such as CPU or I/O bandwidth can become the bottleneck much before the network [2][22]. Anticipating a future shift in DDoS attacks from network to server resources, we explore the vulnerability of Internet applications to sophisticated layer-7 attacks and develop counter-attack mechanisms. In particular, our contributions are (i) classification and experimentation with new application-layer attacks, (ii) development of a mechanism to assign suspicion measures to sessions for scenarios with a potentially small and variable number of requests per session, and (iii) design and experimental evaluation of *DDoS Shield*, a technique that provides DDoS resilience by using suspicion measures and server load to determine if and when to schedule requests to a server.

In studying new classes of attacks, we consider a well-secured system that has defenses against both (1) intrusion attacks, i.e., attacks which exploit software vulnerabilities such as buffer overflows and (2) protocol attacks, i.e., attacks that exploit protocol inconsistencies to render servers inaccessible (e.g., hijacking DNS entries or changing routing). In such a scenario, the only way to launch a successful attack is for attackers to evade detection by being non-intrusive and protocol-compliant, and yet overwhelm the system resources while posing as legitimate clients of the application service. Hence, the only system attributes available for the attacker to exploit are those for the application workload.

We first explore the entire range of exploitable workload parameters and characterize layer-7 resource attacks into three classes: (1) *request flooding attacks* that send application-layer requests at rates higher than for normal sessions; (2) *asymmetric attacks* that send high-workload request types; and (3) *repeated one-shot attacks* in which the attacker spreads its workload across multiple sessions instead of multiple requests per session and initiates sessions at rates higher than normal. For example, an HTTP flood can stress server resources as an asymmetric attack if the attack sessions send requests involving high-computation database queries. We study these classes via testbed measurements of attacks on servers and their hosted web applications. We show that dynamic content presents a substantial heterogeneity in request processing times among request types which can be exploited to initiate asymmetric attacks. While the above attack classes are known to exist for HTTP floods, our work is the first to demonstrate vulnerability to these attack classes for server resources and to implement and compare them experimentally.

Since the attackers mimic legitimate requests, attack sessions are indistinguishable from legitimate sessions via sub-layer-7 techniques. For instance, if the attackers use valid IP addresses from botnets, both server and network attacks would pass undetected by ingress-filtering approaches which check for spoofed source addresses. Further, the server attacks would pass undetected by mechanisms that only detect network anomalies. Thus, we design a comprehensive suspicion assignment mechanism to detect layer-7 misbehavior across the parameters of session arrivals, session request arrivals and session workload profiles. In contrast to traditional anomaly detectors which output binary decisions while bounding the detection and false-positive probabilities, we assign a continuous measure of suspicion to a session which is updated after every request. We establish a set of *soundness principles* that a metric must obey in order to assign suspicion values consistently across workloads with differing numbers of requests per session.

Next, we design a counter-mechanism, *DDoS-Shield*, that uses the suspicion assignment mechanism as an input to a scheduler designed to thwart attack sessions before they overwhelm system resources. The DDoS-resilient scheduler incorporates

S. Ranjan, and E. Knightly are with Rice University, Houston, TX (http://www.ece.rice.edu/networks). R. Swaminathan and M. Uysal are with HP Laboratories, Palo Alto, CA. The research of Ranjan and Knightly was supported by HP Laboratories and NSF Grant ANI-0331620.

the suspicion assigned to a session and the current system workload to decide when and if a session is allowed to forward requests. We develop scheduling policies *Least Suspicion First (LSF)* and *Proportional to Suspicion Share (PSS)* that incorporate suspicion into the scheduling decision. As a baseline for comparison, we implement and study suspicion-agnostic policies such as per-session Round Robin and First Come First Serve among all requests. We also demonstrate the importance of limiting the aggregate rate (over all sessions) at which the scheduler forwards requests to the application system, and we develop an online algorithm to set this rate.

Finally, we effect the three classes of attacks on an experimental testbed hosting an online bookstore implemented using a web server tier, application tier and database tier. We emulate legitimate client workload through an e-commerce benchmark [1]. Using this testbed, we perform a number of experiments to characterize the potency of the attack classes and evaluate the efficacy of DDoS-Shield. Our summary findings are the following:

- Workload asymmetry attacks are more potent compared to request flooding attacks, since they stress the servers significantly more in comparison.
- The repeated one-shot variant of asymmetric attacks are the most potent of the three attack classes due to their ability to get a much larger query flood towards the backend database tier.
- Experimental evaluation of DDoS-Shield indicates that both the scheduling policy and scheduler service rate are critical for an effective counter-DDoS mechanism. The best performance is obtained under the suspicion-aware schedulers, LSF and SPP, when the scheduler service rate is appropriately limited
- Our experiments indicate that 100 legitimate clients that have an average response time of 0.1 seconds under no attack, are delayed to response times of 3, 10 and 40 seconds under the most potent request flooding, asymmetric and repeated one-shot attacks respectively. Furthermore, the efficacy of DDoS-Shield is evident in that the performance under each of these attacks is improved to 0.5, 0.8 and 1.5 seconds respectively.

The remainder of this paper is organized as follows: In Section II, we describe the victim, attacker, and defense models we use to study layer-7 attacks. In Section III, we describe our experimental testbed and characterize the performance impact on legitimate client sessions due to the three attack classes. In Section IV and V we present the design of the suspicion assignment mechanism and DDoS-resilient scheduler respectively and present their experimental evaluation. Finally, we discuss related work in Section VI and conclude in Section VII.

II. Attacker, Victim and Defense System Models

In this section, we (i) describe the attacker model for effecting the protocol-compliant, non-intrusive layer-7 attacks, (ii) present the victim system on which we quantify the performance impact of these attacks and (iii) outline a defense model, DDoS Shield, for detecting and circumventing these new attack classes.

A. Attacker Model

The goal of the attacker is to overwhelm one or more server resources so that the legitimate clients experience high delays or lower throughputs thereby reducing or eliminating the capacity of the servers to its intended clients. The attacker uses the application interface to issue requests that mimic legitimate client requests, but whose only goal is to consume server resources. We assume that the application interface presented by the servers is known (e.g., HTTP, XML, SOAP) or can be readily discovered (e.g., UDDI or WSDL).

We consider session-oriented connections to the server e.g., HTTP/1.1 session on a TCP connection with the server. We assume that the attacker has commandeered a very large number N of machines distributed across a wide-range of geographical areas, organized into server farms popularly known as "botnets." For initiating a TCP session, an attacker can either use the actual IP address of the machine or spoof an address different from any of the addresses in the botnet. Thus, we do not make any assumptions regarding the set of IP addresses accessible by the attacker, and the attacker can potentially use a different IP address for each new session initiated.

We assume that the system has sufficient capacity to support a number of concurrent client sessions much larger than N. Thus, if the attacker were to initiate *normal* sessions concurrently from each of the N machines from the botnet, the system could serve the sessions within acceptable response times.

Using the workload parameters that the attacker can exploit to effect layer-7 attacks, we characterize these attacks into the following three classes:

- **Request Flooding Attack:** Each attack session issues requests at an increased rate as compared to a non-attacking session.
- **Asymmetric Workload Attack:** Each attack session sends a higher proportion of requests that are more taxing for the server in terms of one or more specific resources. The request rate within a session is not necessarily higher than normal. This attack differs from the request-flooding attack in that it causes more damage per request by selectively sending heavier requests. Moreover, this attack can be invoked at a lower request rate, thereby requiring less work of the attacker and making detection increasingly difficult.
- **Repeated One-Shot Attack:** This attack class is a degenerate case of the asymmetric workload attack, where the attacker instead of sending multiple heavy requests per session, sends only one heavy request in a session. Thus, the attacker spreads its workload across multiple sessions instead of across multiple requests in a few sessions. The benefits of spreading are that the attacker is able to evade detection and potential service degradation to the session by closing it immediately after sending the request.

The asymmetric request flooding attack and its variants exploit the heterogeneity in processing times for different request types. The attacker can obtain the information about server resources consumed by different legitimate request types through monitoring and profiling. For this paper, we assume the worst case scenario that the attacker knows the full profiling data, and therefore can select requests such that the amount of server resources consumed per request is maximized. However, in general, this type of information can only be obtained through profiling and timing the server responses from outside. For instance, to obtain the average server processing time per requested page, the attacker uses a web-crawler to obtain the total (network+server) delay in processing a request. and averages to remove the effects of varying loads.

B. Victim Model

We consider a general victim model consisting of a multi-resource pool of servers. In experiments, we focus on an e-commerce application hosted on a web cluster, which consists of multiple-tiers for processing requests, as shown in Figure 1. We define an e-commerce session as an HTTP/1.1 session over a TCP socket connection that is initiated by a client with the web server tier. HTTP/1.1 sessions are persistent connections and allow a client to send requests and retrieve responses from the web-cluster without suffering the overhead of opening a new TCP connection per request. Each request in a session may generate additional processing in the application and the database tiers, depending on the request (or request type). We assume that a request consumes varying amount of resources from each tier (possibly none), consisting of CPU, memory, storage, and network bandwidth. Recall that the goal of the attacker is to push resource usage in one of the tiers to its maximum limit, so that the system capacity for serving clients is diminished.

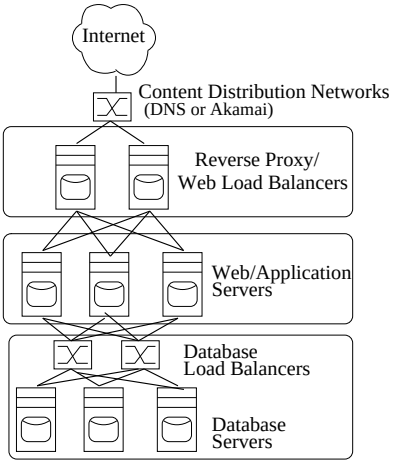


Fig. 1. Victim system model: web cluster hosting a web application.

A legitimate HTTP/1.1 session consists of multiple requests sent during the lifetime of the session. Requests are either sent in a *closed-loop* fashion, i.e., the client sends a request and waits for the response before sending the next request, or they are *pipelined*, i.e., the client could send multiple requests without waiting for their response and thus have more than one request outstanding with the server. A page is typically retrieved by sending one *main request* for the textual content and several *embedded requests* for the image-files embedded within the main page. Main requests are typically *dynamic* and involve processing at the database tier while embedded requests are *static* since they only involve processing at the web-cluster tier.

A client request is processed as follows: First, the client's initial request for a connection is routed by a client-side redirection mechanism such as DNS Round-Robin or Akamai to a reverse-proxy server. The reverse proxy server parses the request's URL and routes the request to a web server typically according to a load-balancing policy (e.g., using round robin or more sophisticated policies as in [6]). If the request is for a static web page or an image file, a server in the web tier serves the requested page. If the request is for an e-commerce functionality, it is served by an application script such as PHP, JSP or Javascript. Such requests typically consist of one or more database queries, the results of which are collated together to produce the response page (*dynamic requests*). Each database query emanating from a dynamic request is forwarded to a database server using a load-balancing strategy [3][22].

Each of the tiers in the system consist of multiple resources: computation, storage and network bandwidth, which are limited in amount. We assume that all tiers continuously monitor the resources in the tier and periodically generate resource utilization reports as well as overall system statistics at the application layer such as throughput and response time. The system is said to be under a resource attack when a surge in a resource usage is accompanied by reduction in throughput and increase in response time without DDoS attack at lower layers.

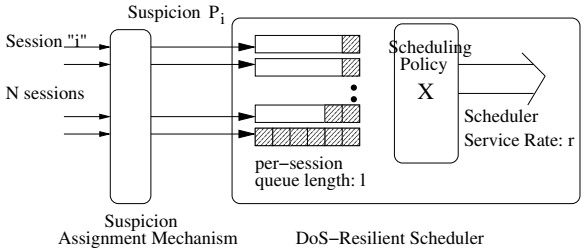


Fig. 2. Defense system model: DDoS-Shield

C. Defense Model

In this paper, we introduce a counter-DDoS mechanism to protect the application from layer-7 DDoS attacks and provide adequate service to legitimate clients even during an attack. The defense model consists of a *DDoS-Shield* which is integrated into the reverse-proxy and thus intercepts attack requests from reaching the web-cluster tiers behind the reverse-proxy. The DDoS-Shield examines requests belonging to every session, parses them to obtain the request type and maintains the workload- and arrival-history of requests in the session. Figure 2 shows the system architecture for DDoS-Shield that consists of: (1) Suspicion assignment mechanism which uses the session history to assign a suspicion measure to every client session i as described in Section IV; and (2) DDoS-resilient scheduler that decides *which* sessions are allowed to forward requests and *when* depending on the scheduling policy and the scheduler service rate, as discussed further in Section V.

III. Vulnerability to Attacks

In this section, we characterize the effectiveness of the layer-7 DDoS attacks in overwhelming the server resources on our e-commerce application. We first quantify the variation in processing times for different requests and then mount each of the three classes of layer-7 DDoS attacks to demonstrate the potency of each attack class.

A. E-Commerce Testbed

The example e-commerce application that we consider is an online bookstore hosted on a multi-tiered architecture consisting of three web servers and one database server. We use Apache to implement the web server, PHP scripting to implement the application logic, and MySQL to implement the database server. The networking infrastructure consists of 100 Mbps links for

both the access links to the system and for the connections between tiers. The servers are Intel Pentium IV 2.0 GHz processor machines running Linux 2.4.18 kernel with 512 MB SDRAM and a 30 GB ATA-66 disk drive.

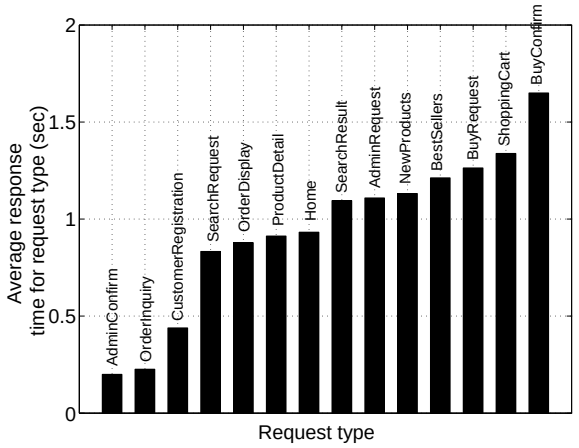


Fig. 3. Heterogeneity in processing times for different dynamic content requests in online bookstore application.

Recall that the effectiveness of an asymmetric workload attack arises from large differences in processing times of different request types. To explore whether this is possible for our online bookstore implementation, we profiled the processing times of individual request types to identify requests with high resource consumption on the server. Figure 3 shows the response times perceived across different types of requests for the online-bookstore application on our experimental system. We note that the most expensive request is about 8 times more expensive than the least. Expensive request types such as "BestSellers" involve heavy CPU processing on the database server since they initiate queries that involve table join operations across multiple tables followed by a sort operation to obtain a list of top-selling books.

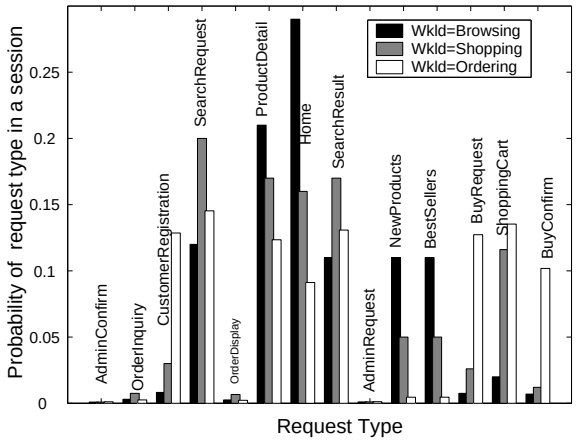


Fig. 4. Probability of occurence of a request type in a client session for browsing, shopping and ordering sessions. Browsing sessions send only 5% requests for pages that involve write queries to the database server, while shopping and ordering sessions send an increasing percentage of such requests.

Next, we attempt to quantify the *potency* of various layer-7 DoS attacks in our system. We use the following metrics to measure the potency of an attack: (1) CPU utilization on the web and database tiers – the main resource being attacked in our experiments; (2) average response time of requests as an indication of the slow down a legitimate client will experience; and (3) average throughput in requests/second achieved per normal client session. We also quantify the ease of mounting a layer-7 DoS attack at the attacker end point by: (1) the number of unique IP addresses required and (2) the aggregate bandwidth needed to launch the attack.

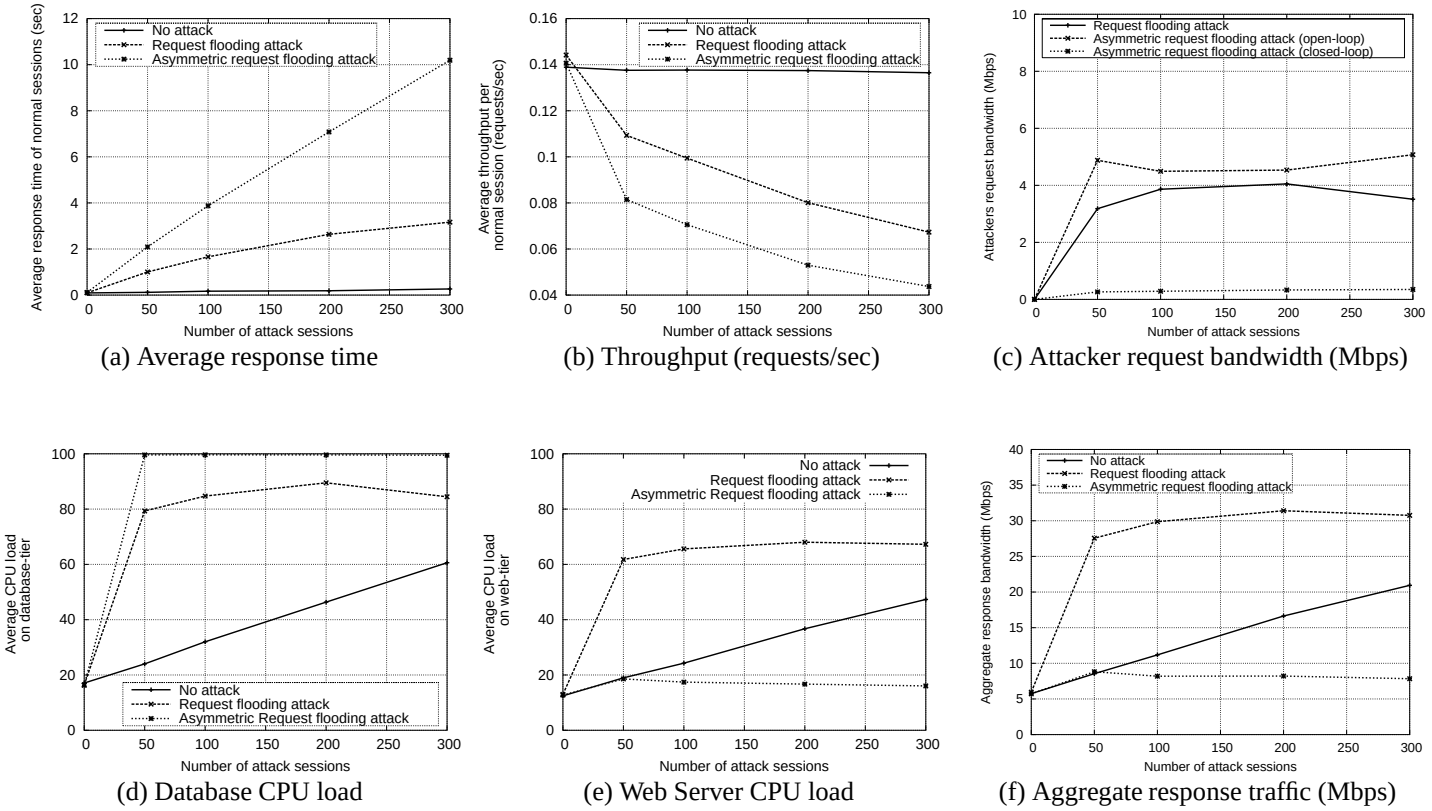
We emulate the workload of a legitimate client session using the session types shown in Figure 4 based on the TPC-W benchmark [1]. In particular, in each experiment, we use 100 HTTP/1.1 sessions, 33% in each of browsing, shopping and ordering profiles, to represent the legitimate client population. Legitimate clients generate new sessions using an exponential distribution with mean of 0.2 seconds. Requests are submitted to the web servers using exponentially distributed think times with a mean of 7 seconds between receiving a response and issuing the next request.

We generate each of the three types of attacks as follows: First, the request flooding attack is mounted by decreasing the think-times between requests to values lower than the normal 7 seconds. For maximal potency, we decrease the think-times to 0, thereby, generating the requests as fast as possible. Second, the asymmetric workload attack is generated using one of the expensive request types, BestSellers. We mount this attack with the normal think-time of 7 seconds between requests first, and then combine it with the request flooding attack by reducing the think-times to 0. For each experiment involving request flooding or asymmetric request flooding attacks, we vary the number of attack sessions from 0 to 300 sessions to simulate "no attack" and "large attack" scenarios respectively. Finally, the repeated one-shot attack is mounted by repeatedly generating single request sessions for the BestSellers script using inter-arrival time between sessions smaller than the legitimate mean 0.2 seconds. Once the response to the single request is received by the attacker, it closes the session and creates a new one.

B. Attack Potency

Figure 5 shows the results from the experiments designed to quantify the potency of each type of attack. Our results indicate that the response time of normal sessions increases from 0.1 seconds under no attack to as high as 3 and 10 seconds when there are 300 attack sessions in the request flooding and asymmetric request-flooding attacks respectively. Thus, assuming that user patience for web page download times is 5 seconds [7], an asymmetric attack would also drive legitimate users away from the web site. Furthermore, the throughput of each normal session in terms of requests completed per second per session also drops drastically from 0.14 to 0.065 and 0.042 under request flooding and asymmetric request-flooding attacks respectively. Moreover, the repeated one-shot attack is much more potent than any other attack class as seen from Figure 6(c). In the most potent form of the attack, when the attacker waits 0 seconds between closing and opening another session, the average response time per normal client session increases to as high as 40 seconds.

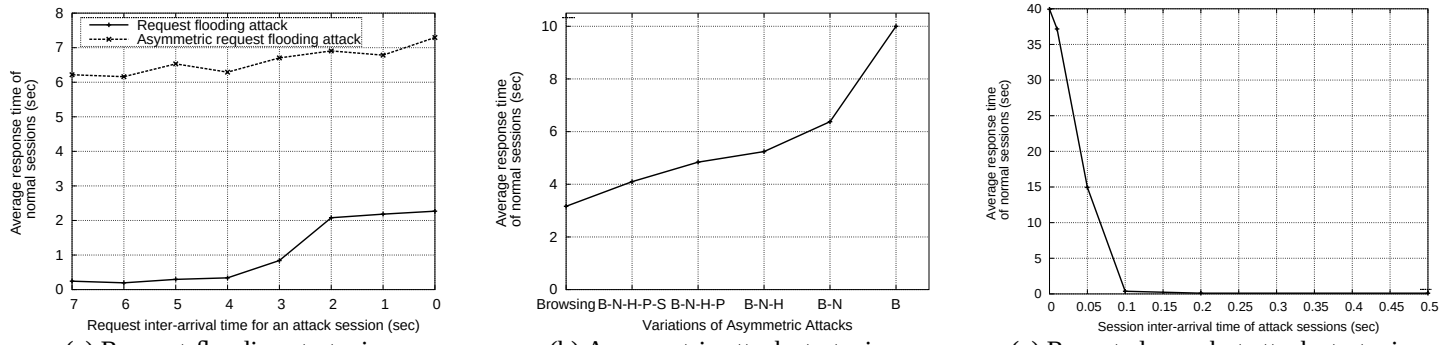
Both repeated one-shot and asymmetric attacks make the database server CPU the bottleneck, driving the CPU loads to almost 100%, in their most potent forms. However, the asymmetric attack is limited in sending a query flood towards the backend database server since the web server serves only one



(a) Average response time (b) Throughput (requests/sec) (c) Attacker request bandwidth (Mbps)

(d) Database CPU load (e) Web Server CPU load (f) Aggregate response traffic (Mbps)

Fig. 5. Effect of most potent request flooding attack (attacker think-time=0 sec) and asymmetric request-flooding attack (attacker uses "BestSellers" script) on 100 normal sessions.



(a) Request-flooding strategies (b) Asymmetric attack strategies (c) Repeated one-shot attack strategies

Fig. 6. Variations in attacker strategies. The figures show the performance impact on 100 normal sessions. In (a), the attacker uses 200 attack sessions to launch a request flooding (browsing profile) and asymmetric request flooding (BestSellers script) attack, while varying the request inter-arrival times as $[0-7]$ seconds. In (b), the attacker uses 300 attack sessions sending requests as fast as possible, while varying the attack workload. In (c), the attacker uses only one session at a time, sending one BestSellers request per session and varies the inter-session time from $[0-0.5]$ seconds.

request at a time per session. In contrast, the repeated one-shot attack is successful in sending a larger query flood towards the database server, since after being blocked on a session, the attacker opens yet another session and sends another request which translates into more queries towards the database server. This query flood leads to much higher queuing delays at the database server which explains the higher potency for repeated one-shot attacks. Figure 7 depicts the inter-arrival times between queries received at the database server. The figure shows that 90% of queries arrive within 10 msec of the previous query for the repeated one-shot attack, compared to the 80% for the asymmetric attack. Moreover:

- Asymmetric request-flooding is significantly more potent than normal request flooding attack since it succeeds in making the database CPU the bottleneck, as observed from Figure 5(d). In contrast, the normal request flooding attack never makes the database CPU the bottleneck and only succeeds in increasing the web server CPU loads to as high as 70%. Since, in the online-bookstore implementation, the database server is more sensitive to heavy loads than the web server, the asymmetric request-flooding attack delays normal sessions significantly more.
- All attacks are server attacks and the network-access link to

the cluster (100 Mbps) is never overwhelmed as observed from Figure 5(f). The reason that the aggregate download traffic saturates much before 100 Mbps is that the web or database server CPUs are overwhelmed. Since asymmetric attacks bottleneck at the database servers, the download traffic is much less at 8 Mbps compared to 31 Mbps for request flooding attacks.

- The increase in response times is not caused by the system being overloaded due to too many client sessions; the slowdowns are directly attributable to the system doing more work as a response to attack requests. Observe that the response times for the normal sessions are almost constant at 100 msec when the attackers behave exactly like normal sessions, i.e., have the same workload as well as think-time profiles.
- The asymmetric workload attack is a low-rate attack, since it requires a lesser number of attack sessions to inflict damage of similar magnitude. Also, all attacks are quite easy to implement since (1) they require access to approximately 300 unique IP-addresses, easily obtainable using current-day server farms or botnets and (2) the maximum aggregate bandwidth needed to launch an attack is 5 Mbps upstream for requests and 26 Mbps downstream for response traffic, easily achievable using current-day access networks.
- Changing the baseline normal client workload from 100 client sessions causes a corresponding linear change in the number of attack sessions needed to cause similar damage.

Finally, while it may appear that an attack that pipelines requests without waiting for their responses would cause more damage than the attack which sends its requests in a closed-loop. However, Apache web servers only service one request per session at a time. Hence, even though an attack session may send multiple requests, they end up waiting in Apache's per-session queue, until Apache has completely serviced the last request, which may involve sending database queries and receiving their responses. As a result, attackers that generate requests in an open-loop without waiting for the responses to arrive are only slightly more effective than closed-loop attack sessions. Moreover, these open loop attacks are higher rate attacks and hence easily detectable compared to closed loop attacks. Observe from Figure 5(c), than an asymmetric open loop attack sends ≈ 4 Mbps of request traffic compared to the much lower 0.4 Mbps by an equivalent closed loop attack, for similar damage.

C. Attacker Strategies

Since the most potent attacks are also the most deviant from normal behavior and hence most easily detectable, the attacker may employ lower-potency attacks to evade detection and hence guarantee success. Next, we assess the damage caused by these lower-potency attacks.

- *Variable request-arrival rate:* Instead of sending requests as fast as possible (attack think-time=0 sec), the attacker decreases its request-rate. Observe from Figure 6(a), that the asymmetric request-flooding attack still causes similar damage to the normal sessions even when the attack sessions send requests at periods as large as 7 seconds. This validates our hypothesis that asymmetric attacks are more potent due to their workload-asymmetry rather than rate-asymmetry.
- *Variable session workload:* Instead of sending only the heaviest BestSellers requests in a session, the attacker morphs its sessions into profiles increasingly similar to the normal profiles. Thus, with reference to Figure 3, suppose an attacker selects

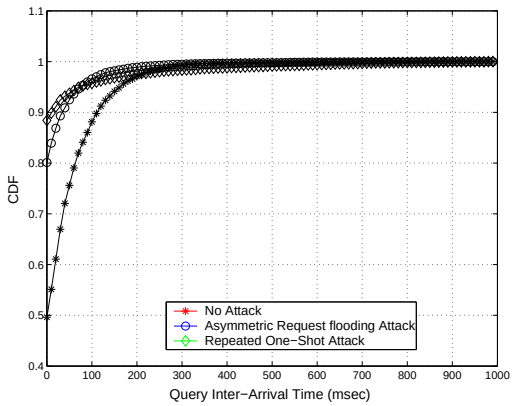


Fig. 7. CDF for inter-arrival times for query-arrivals at the database server for different attacks. The number of normal browsing sessions is 100 in each scenario. The no attack case has 0 attackers, while the asymmetric request-flooding corresponds to 300 attack sessions with request think-time of 0 seconds. The repeated one-shot attack corresponds to an attack session being opened immediately (0) seconds after closing the previous session.

the following request types in decreasing order of processing times: **B**estSellers > **N**ewProducts > **H**ome > **P**roductDetail > **S**earch. We investigate the following attacker strategies: (1) B: 100% BestSellers requests, (2) B-N: equal number of BestSellers and NewProducts requests and similarily, (3) B-N-H, (4) B-N-H-P, (5) B-N-H-P-S. Figure 6(b) shows the damage caused to 100 normal sessions by 300 attack sessions. In each experiment, the attack sessions send requests as fast as possible using one of the workload profiles mentioned above. As observed, the damage decreases consistently as the attack sessions dilute the proportion of the heaviest BestSellers requests, approaching the potency of the normal request flooding attacks which have the same workload profile as the legitimate clients.

- *Variable inter-session arrival time:* In the repeated one-shot attack, the attacker may emulate slower inter-session rates by increasing the waiting time between closing and opening the next session. Figure 6(c) shows that the attack potency decreases consistently with increasing inter-session time between attack sessions. Furthermore, when the attack session uses the same inter-arrival time as normal sessions (0.2 seconds), there is no performance degradation.

IV. Quantifying Attack Suspicion

Because attackers cannot be distinguished from non-malicious clients with 100% certitude, our objective is to provide a mechanism to tag each session with a continuous measure of suspicion. In our architecture, this value is then used by a request scheduler to determine when and if to service a particular request.

We formulate the suspicion-assignment problem by first performing measurements to characterize the set of distributions that define legitimate behavior. We then calculate the suspicion of a session on the basis of the probability that it was generated from one of the legitimate distributions. Recall from Section III that attacks succeed by altering either of the session parameters of *session inter-arrival time*, *request inter-arrival time* or *session workload-profile*. Thus, we design suspicion assignment techniques to assign a suspicion measure to a session with respect to each of these parameters. These individual values are then combined into one suspicion measure for the session.

First, we describe an offline phase in which we build legitimate client behavior profiles using system logs, which are assumed to be un-influenced by attacks. Next, we describe suspicion assignment techniques corresponding to each of the three kinds of deviations from normal behavior, followed by an algorithm to combine their outputs. Finally, we conclude by presenting testbed results to evaluate the performance of our suspicion assignment techniques.

A. Legitimate Client Profiles

In this phase, we extract information from system logs to build profiles for legitimate client behavior with respect to session inter-arrival times, request inter-arrival times, and session workload profile. The system logs store the number of requests per session and the resources consumed by a request for each of the resources: CPU, disk and network bandwidth. We assume stationarity in the system logs.[1]

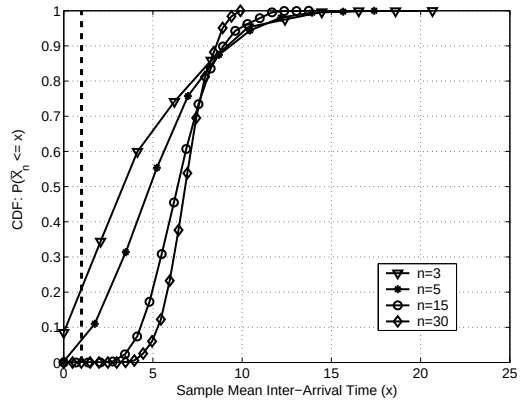


Fig. 8. Cumulative Distribution Function for request inter-arrival time: $P(\overline{X_n} \leq x)$ with varying session lengths or sample sizes 'n'.

• **Session Inter-Arrival Distribution:** We extract the aggregate session inter-arrival times to obtain the empirical distribution A. Due to our workload generator, A is exponential with mean 0.2 seconds.

• **Request Inter-Arrival Distribution:** First, we extract sessions of length equal to n requests. Next, we use the request inter-arrival times of these sessions and obtain an empirical distribution of request inter-arrival times: X_n. This is done for all values of $n = [2, 60]$ as shown in Figure 8. With increasing sample size n, X_n tends to an exponential distribution with mean of 7 seconds, corresponding to the distribution used in our workload generator.

• **Session Workload Profile:** Using the resource consumption for a request for each resource type (CPU, disk, network), we use a standard centroid-clustering algorithm [8][19] to obtain workload profiles for legitimate sessions as follows: First, requests with similar resource consumption for a resource are grouped into several *request-resource classes*. Next, sessions with a similar proportion of requests per request-resource class are grouped into several *session types*.

Recall that in our system, the attacks overwhelm the CPU resources on the database tier. Hence, we extract the database CPU clock cycles consumed by each request from the logs and cluster requests with similar CPU utilization. Starting with all the requests from the logs, each defining its own cluster, we group requests with similar CPU utilization at every iteration to obtain a decreasing number of clusters. This is done until a normalized ratio between the inter- and intra-cluster distances [8] reaches a local maxima, thus obtaining a set of r request types: $\cup_{i=1}^{r}\{a_i\}$; identified by their average CPU utilization. Similarly, sessions defined as a histogram on the set $\cup_{i=1}^{r}\{a_i\}$ of request classes, are clustered to obtain an optimal set of c session types: $\cup_{j=1}^{c}\{G_j\}$.

In our example online-bookstore implementation, the clustering algorithm groups requests into 14 request classes. Incidentally, each of these request classes also corresponds to a particular type of page that was being requested, e.g., *Home* and *Best-Sellers*. Sessions are clustered into 3 session types, identified as: *browsing*, *shopping* and *ordering* (see Figure 4).

B. Detection of Session Arrival Misbehavior

Recall that a repeated one-shot attack's potency is due to the higher than normal session arrival rates. Hence, detection of these attacks is based on detecting increases in session inter-arrival times. Upon the arrival of a new session i, we first calculate the difference between its arrival time and that of the last session: α_i. Then, using the distribution A for legitimate session arrival times, we assign it a seed suspicion $f_{session}(i)$ as the probability that we would have observed a session inter-arrival time less than α_i: $f_{session}(i) = 1 - P(A \leq \alpha_i)$.

This method has high false positives since a legitimate session that arrives in between two consecutive one-shot sessions would also be assigned a high seed suspicion. However, in the latter half of this section, we present an algorithm to reduce the performance impact due to these false positives.

C. Detection of Request Arrival Misbehavior

A request flooding attack succeeds by sending requests at rates higher than normal. Hence, detection of these attacks is based on detecting decreases in inter-arrival time between successive requests in a session.

On observing the nth request in a session, we assign its suspicion as follows: (1) calculate the mean inter-arrival time μ over n requests seen in the session so far; (2) use sample distribution $\overline{X_n}$ shown in Figure 8 to assign suspicion as: $f_{request}(i) = 1 - P(\overline{X_n} \leq \mu)$. Thus, the suspicion measure for an attack-session which sends requests once every 1 seconds would be 0.8 after 5 observations, quickly increasing to 0.92 after 10 observations.

D. Detection of Session Workload Misbehavior

Recall that in asymmetric attacks, the attacker exploits heterogeneity in the server processing times of requests and selectively sends more requests towards the heavy request classes. Thus, a system under attack would see sessions with a higher than normal proportion of requests for certain request classes. Hence, detection of asymmetric attacks is based on detecting deviations in the workload profile of sessions.

Given, a set $\cup_{j=1}^{c}\{G_j\}$ of c ideal session types, detection of workload misbehavior is formulated as an *online estimation* of the probability that the requests belonging to a session is distributed as one of the legitimate or ideal session types G_j. Initially, we assume there is only one ideal session-type G. Due to

[1] This assumption can be relaxed and time-or-day effects can be incorporated using standard techniques from time-series analysis.

the discrete number of request types, an equivalent problem is observing a series of throws of a dice with r faces and generating distribution G, and estimating whether the observed series is generated from the distribution G.

Given an ideal session type G, a suspicion measure assigns suspicion numbers to a session s by using (1) the length of the session n and (2) the deviation d of the session from ideal behavior as captured by a *distance metric* between the session type and the ideal type. Next, we develop a framework for *soundness* of a workload suspicion measure to ensure consistency in assignment of suspicions across workload deviations.

A desirable distance metric disassociates session length (corresponding to the number of request observations) from deviation and assigns sessions which have the same deviation from ideal type, an equal distance, irrespective of their lengths. The other properties that we desire in a distance metric are that distance grows with deviation from the ideal type and distance between a type and itself is 0. In this paper, we consider two candidate distance metrics to illustrate the properties: the *Kullback Leibler (KL)* distance metric [10] and a metric we developed, which we call the *Residue Factor (RF)* metric.

Let $\sigma = \cup_{i=1}^{r}\{a_i\}$ denote the set of r request classes. Denote a session s as a histogram on the number of requests $n(a_i)$ seen per request class: $s = \cup_{i=1}^{r}\{n(a_i)\}$. Similarly, define a session type $T(s)$ as a histogram on the fraction of requests $N(a_i) = \frac{n(a_i)}{n}$ seen per request type: $T(s) = \cup_{i=1}^{r}\{N(a_i)\}$; where n is the total number of requests seen in the session: $n = \sum_{i=1}^{r} n(a_i)$. Further, define the ideal session type $G = \cup_{i=1}^{m}\{G(a_i)\}$, where $G(a_i)$ denotes the fraction of requests of request type a_i and $\sum_{i=1}^{r} G(a_i) = 1$.

D.1 Distance Metrics

Definition 1: The KL distance between the session type $T(s)$ characterizing a session s and the ideal distribution G is defined as:

$$KL(T(s)||G) = \sum_{a_i \in \sigma} N(a_i) \log \frac{N(a_i)}{G(a_i)}. \tag{1}$$

Next, we define a Residue Factor (RF) distance by extracting the *g*reatest *c*ommon *f*actor (gcf) of G present in session s: $gcf = \min_i \lfloor n(a_i)/G(a_i) \rfloor$. Now, define residue $res = \sum_{i=1}^{r}\{n(a_i) - gcf\, G(a_i)\}$.

Definition 2: The RF-distance metric between a session s and ideal type G is defined as:

$$RF(s||G) = \frac{res}{gcf} \tag{2}$$

Intuitively, the greatest common factor and residue represent the subtypes within session s that are good and bad with respect to the type G. Hence, the RF-distance penalizes a type for deviating away from G (as captured by the residue) while rewarding it in proportion to the length for which it was well-behaved (as captured by gcf). We handle the case where there is no good subtype, i.e., $gcf = 0$ separately.

Observe that both KL-distance and RF-distance have the properties desirable in a distance metric. We illustrate with an example on two request classes: $\sigma = \{0, 1\}$ and a Bernoulli ideal distribution G having probabilities (0.5,0.5). If a session s has the same type as the most likely realization of G, then both distance metrics assign distance 0. In contrast, sessions originating from a Bernoulli $(0.8, 0.2)$ distribution and having types such as $(4, 1), (8, 2)...k(0.8, 0.2)$ are assigned KL-distance of 0.193 and RF-distance of 1.5, irrespective of length. Moreover, their distance is less (on average) than that assigned to sessions originating from Bernoulli $(0.9, 0.1)$ distributions, in which case the average KL-distance is 0.368 and RF-distance is 4.

D.2 Soundness

A suspicion measure f is said to be *sound*, and hence consistent in assigning suspicion across workload misbehavior, if it obeys the following properties:

- **Zero-Distance Property**: A session s with the same type as the ideal session type is always assigned a suspicion number of 0, irrespective of its length n.

$$T(s) = G \implies f(s) = 0 \;\; \forall n \in [1, \infty) \tag{3}$$

That is, if a session has the same type as the ideal, its deviation from the ideal type is 0, and hence its suspicion is 0.

- **Distance-Proportionality Property**: Amongst all sessions of the same length n, a session which deviates more from the ideal session type is assigned a higher suspicion. Thus, given two sessions s_1 and s_2 of lengths n_1 and n_2 and distances from ideal type d_1 and d_2 respectively:

$$n_1 = n_2, \;\; d_1 > d_2 \implies f(s_1) > f(s_2) \tag{4}$$

That is, greater deviation from the ideal type signifies greater suspicion.

- **Length-Proportionality Property**: If two sessions have the same type which is different from the ideal type, then the session with greater length is assigned higher suspicion. Thus, given two sessions s_1 and s_2 of lengths n_1 and n_2 and distances from ideal type d_1 and d_2 respectively:

$$T(s_1) = T(s_2) \neq G, \;\; n_1 > n_2 \implies f(s_1) > f(s_2) \tag{5}$$

That is, with an increased number of observations, the suspicion probability converges towards its true value.

There are several possible measures f which satisfy the properties of soundness. We next consider a class of suspicion measures which are derived directly from the properties of soundness, and hence correct, while also being computationally efficient.

D.3 Length Distance Product (LDP) Measure

Definition 3: Define a Length Distance Product (LDP) measure as one which assigns suspicion to a session s of type $T(s)$ as the product of its length and distance from the ideal type G. Substituting by the two distances of KL-distance and RF-distance considered in this paper, we have the following equivalent measure definitions:

$$\begin{aligned} f_L^{KL}(s) &= n\, KL((T(s)||G) \\ f_L^{RF}(s) &= n\, RF(S||G) \end{aligned} \tag{6}$$

If there are multiple ideal distributions or types: $\cup_{j=1}^{c}\{G_j\}$, each of them equally likely then the LDP measure is defined with respect to the distribution which is the closest in terms of distance:

$$f_L^{KL}(s) = n \min_j(KL(\,(T(s)||G_j)\,))$$
$$f_L^{RF}(s) = n \min_j(RF(\,(s||G_j)\,)) \quad (7)$$

Since by definition the LDP measures are proportional to distance and length, it is easy to see that they obey all the properties of soundness. The suspicion values assigned by LDP measures are no longer contained within 0 and 1. However, it is relatively straightforward to do so by choosing a very large number N and normalizing such that

$$f_L(s) \leq N \implies f_L(s) = f_L(s)/N$$
$$f_L(s) > N \implies f_L(s) = 1.0$$

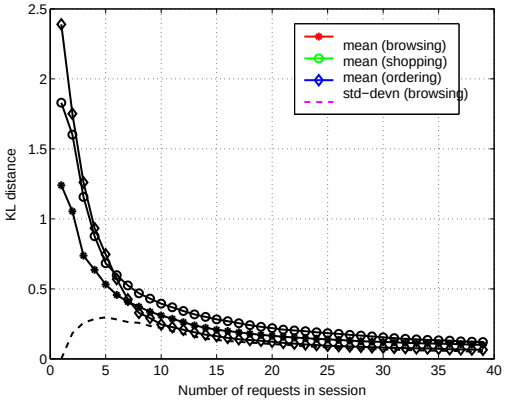


Fig. 9. Mean of KL-distance of "browsing", "shopping" and "ordering" sessions with increasing sample-sizes n.

Our online bookstore implementation consists of three ideal distributions: $G_{browsing}$, $G_{shopping}$ and $G_{ordering}$ as shown in Figure 4. Figure 9 shows the average KL-distance of a browsing session with respect to $G_{browsing}$ with increasing number of requests n. Note that the KL-distance of a legitimate session with its ideal distribution converges to 0 with increasing number of requests n.

D.4 Assignment of Net Suspicion

We next describe an algorithm to aggregate the suspicion measures across the various misbehaviors into one suspicion measure per session. Given a session s, denote the seed suspicion that was assigned to this session on its arrival by $f_{session}(s)$. As the session proceeds in sending requests, after observing n requests, it is assigned a suspicion measure by each of the request arrival and workload misbehavior detectors as: $f_{request}^n(s)$ and $f_L^n(s)$. Thus, using a suspicion weighting parameter $0 \leq \beta \leq 1$, we define the net suspicion measure $f^n(s)$ as follows:

$$f^n(s) = f_{session}(s) * (\beta\, f_L^n(s) + (1-\beta)\, f_{request}^n(s)) \quad (8)$$

Note that net suspicion is within 0 and 1, and has the following desirable features:

- As discussed earlier, there is a high false-alarm rate in the session arrival misbehavior detector, and hence legitimate sessions which get caught between successive one-shot attack sessions may be flagged with a high seed suspicion. Hence, if the session was really legitimate, then it would obey the workload and request-arrival profiles and hence would get a chance to improve its suspicion. In contrast, if the session is part of a repeated one-shot attack then it will be given a high seed suspicion enabling the system to service it with lower priority.
- The suspicion of a session with respect to workload- or request-arrival suspicion is weighted by the parameter $0 \leq \beta \leq 1$, which is set depending on which of the two suspicion measures has potential for greater damage to the system. We illustrate with an example: consider two sessions i and j with suspicion probabilities $(f_L, f_{request})$ as $(0.2, 0.8)$ and $(0.8, 0.2)$ respectively. If workload-misbehavior is considered more potent, then weighing them with $\beta > 0.5$ would consider session i more suspicious. Similarly, if request misbehavior is considered more potent, then setting $\beta < 0.5$ would consider session j as more suspicious. We chose to weigh both misbehaviors equally, and hence $\beta = 0.5$ in our system.

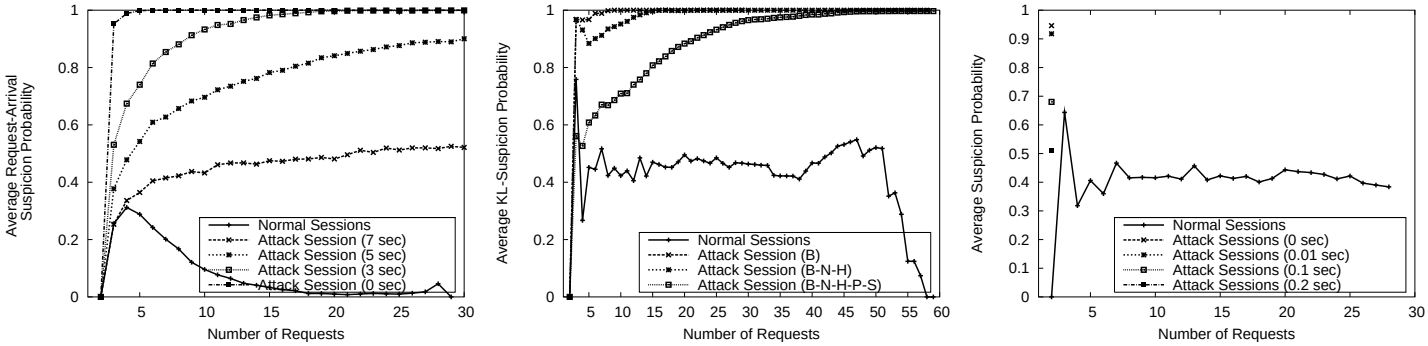
D.5 Performance of Suspicion Assignment

We next provide numerical results for the performance of the suspicion assignment techniques on attacks launched against the online-bookstore implementation. Figure 10 shows the behavior of suspicion measure with an increasing number of requests in a session. Notice that the scheme obeys the properties of soundness in that the suspicion of a session either converges to 0 or 1 with more observations, depending on whether the session is legitimate or malicious. We make the following observations:

- In either request flooding or asymmetric attacks, the attack sessions can be distinguished from normal sessions after 4 requests on average enabling counter-DDoS mechanisms to quickly punish attack sessions.
- Normal sessions converge to suspicion of 0 with respect to request-arrival and workload after 17 and 57 requests respectively as seen from Figure 10(a),(b).
- A request flooding attack session sending requests at think-times of 0 seconds is detected with certitude of 1.0 after 5 requests on an average. Moreover, the lower the attack rate i.e., the higher the value of think-time used by an attack session, the more observations needed to detect it with certainty.
- An asymmetric attack session sending BestSeller requests is detected with suspicion probability 1.0 after 8 requests on average. Moreover, if the attacker morphs its identity by mixing other request types in an attack session, then the number of observations needed to detect the attack session with certainty increases.
- Attack sessions involved in repeated one-shot attack of highest potency (inter-session time=0) are assigned seed suspicion of 0.95 on an average. Normal sessions also start with similar seed suspicions but the effect of high initial suspicion is diluted by the lower suspicions assigned to them by the request-arrival and workload suspicion assignments.

V. Scheduler Design for DDoS-Shield

In this section, we present the DDoS-resilient scheduling policy of *DDoS-Shield*, which combines the continuous measure of suspicion assigned by our suspicion mechanism with the cur-



(a) Request flooding attacks (b) Asymmetric attacks (c) Repeated one-shot attacks

Fig. 10. Average suspicion probability of a normal or attack session with increasing number of requests seen in the session.

rent system workload to decide whether and when a session is allowed to forward requests (see Figure 2). The DDoS-resilient scheduler is integrated into the reverse proxy, and can thus intercept requests belonging to malicious sessions before they overwhelm system resources.

A. Aggregate Scheduling Rate and Eligibility

The maximum aggregate rate at which the scheduler forwards requests to the web cluster is a configurable parameter termed the *scheduler service rate*: r requests/second. We will show that a wide range of rates yields high performance. However, setting the rate too low results in an under-utilized backend whereas setting it too high results in all requests (including malicious ones) being sent to the backend.

Each session has a backlog queue for requests which haven't been forwarded to the web cluster, and requests are dropped using a Drop-Tail policy when the length of the queue exceeds a configurable parameter *per-session queue length*: l. At rate r, the scheduler picks a session from amongst the eligible sessions and forwards its Head-of-Line (HoL) request to the web cluster.

We determine the eligibility criterion for a session by allowing only one outstanding *main request* per session. Recall that main requests are typically requests for the dynamic page and are followed by embedded requests for static content, typically image files that are embedded in the page. Thus, a session is considered eligible for scheduling only if its last main request has been serviced by the web cluster and the response sent to the client. This is in agreement with the behavior of the Apache web server, which also services only request per session at any time.

Figure 11 shows the state diagram for a session in the scheduler queue. A new session starts in the state *Allow All* and once it is scheduled by the scheduler, its main request is forwarded to the web tier, after which the session's state is changed to *Allow Embedded-Only*. In accordance with the HTTP/1.1 specification for pipelining, any embedded requests sent by the client are forwarded to the web tier, irrespective of whether the main request has been serviced or not. However, if we receive another main request, it is kept waiting in the session queue.[2] On receiving the response for the main request, the session is made eligible for being scheduled again, by changing its state to *Allow All*, after which the HoL main request in this session's queue can be forwarded when the session is scheduled again.

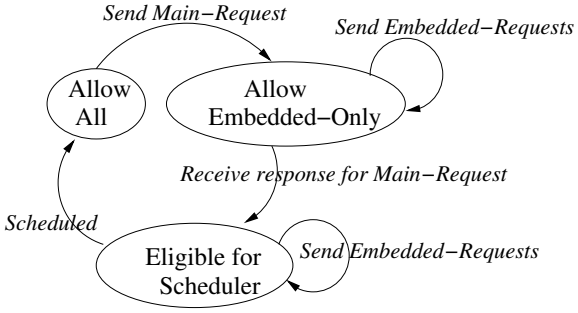


Fig. 11. State Diagram for a session in scheduler queue

B. Scheduling Policies

We introduce the following scheduling policies to protect the system from DDoS attack:

- **Lowest Suspicion First (LSF) Scheduler:** The *cost-optimal* scheduler is one which obtains a schedule such that for the N eligible sessions in the system at any time, each with suspicion probability as p_i, their average response time d_i realizes the following objective function:

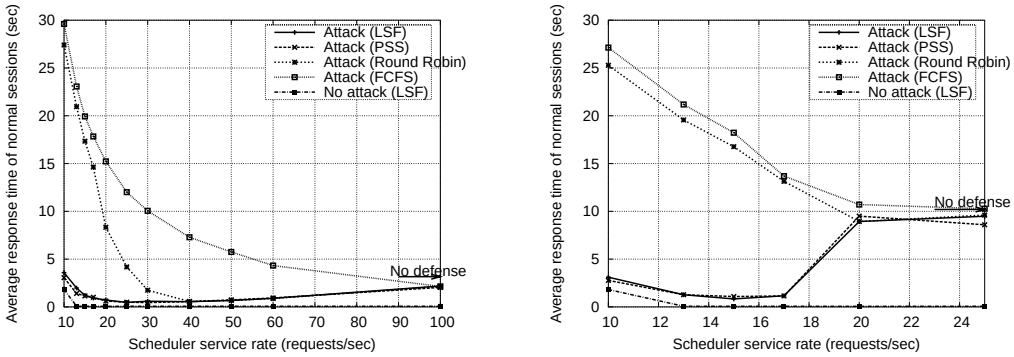
$$\min \sum_{i=1}^{N} (1 - p_i)(d_i) \tag{9}$$

Intuitively, this objective function maximizes the sum total of suspicion probabilities (p_i) for requests queued at the DDoS scheduler so that those with low suspicion are forwarded to the web cluster. Thus, the cost-optimal scheduler is a strict-priority scheduler which selects the top sessions after sorting them in decreasing order as: $(1 - p_1) \geq (1 - p_2) \cdots \geq (1 - p_N)$.

- **Proportional to Suspicion Share (PSS) Scheduler:** The cost-optimal scheduler may result in starving sessions with high suspicion probability p_i. Hence we also design a *max-min fair* algorithm with the fairness objective of assigning forwarding-rates r_i to sessions in proportion to their confidence probabilities $1 - p_i$:

$$\frac{r_i}{r_j} = \frac{1 - p_i}{1 - p_j} \tag{10}$$

[2]Hence, w.r.t. a client which sends pipelined main requests, we are still HTTP/1.1 compliant, in that requests now wait in the reverse proxy server queue instead of Apache queue.



(a) Request flooding attack (b) Asymmetric attack

Fig. 12. Effect of various scheduling policies and scheduler service rates on 100 normal sessions. (a) shows performance under most-potent 300 request flooding sessions while (b),(c) shows performance under most-potent 300 asymmetric attack sessions.

As a baseline for comparison, we also implement two schedulers that are agnostic to suspicion measures: (i) First-Come First-Serve (FCFS) which schedules the session with the earliest arrived HoL request from amongst all the eligible sessions and (ii) Round Robin which schedules requests one per session in round-robin order only among eligible sessions.

Next, we propose an online algorithm to set the *scheduler service rate* r of the scheduler as a function of the sum of confidence probabilities of the active sessions at any time. Assume there are N eligible sessions at the scheduler at time t. Denote the $95\%ile$ of the throughput in terms of completed requests/second achieved by a legitimate session under no attacks as $r_{0.95}$. The scheduler service rate r is adjusted every $update\_interval$ using an Exponential Weighted Moving Average function: $r = \alpha * r + (1-\alpha) * r_{new}$. The rate r_{new} is the sum of the individual session-rates: $r_{new} = \sum_i^N r_i$, each of which is obtained as a linear function of the session's suspicion probability as follows: $r_i = (1-p_i) r_{0.95}$.

C. Performance Evaluation

First, we establish through experiments that to be effective, a counter-DDoS mechanism needs both scheduling and a properly set aggregate service rate. We compare the suspicion-aware scheduling policies, LSF and PSS against the suspicion-agnostic policies, Round Robin and FCFS. The per-session queue length is fixed at 100 requests. We also compare the performance of the scheduling algorithms against two baseline scenarios: (1) *No Attack* when there are 0 attack sessions; and (2) *No Defense* when all the attack sessions are present but no defense strategy is used, i.e., the scheduler is FCFS with per-session queue lengths set to infinity.

Figure 12 depicts the average response time for normal sessions as a function of the maximum rate that the scheduler forwards the aggregate. Performance for the two attacks (Figures 12(a) and 12(b)) is discussed below.

C.1 Request-flooding Attack

We first consider the most potent request-flooding attack using 300 attack sessions on 100 normal sessions in Figure 12(a).

- The strategy of using both scheduling and limiting the aggregate service rate is critical to achieving DDoS resilience. The best performance is obtained on using a LSF or PSS scheduler with scheduler service rate set in the range of 15 to 50 requests/second.
- DDoS-Shield is effective in thwarting the request flooding attack, as evident from the fact that performance improves to 0.5 seconds from the 3 seconds under no defense. Further, note that there is minimal penalty due to false positives (legitimate sessions being delayed) or, false negatives (malicious sessions being admitted).
- The LSF and PSS schedulers perform the best, with LSF slightly better. The Round-Robin and FCFS schedulers are agnostic to suspicion probabilities and still admit many malicious sessions leading to significantly lower performance. Round-Robin is still better than FCFS since in comparison it schedules more non-attacking sessions in every round.
- Performance is non-monotonic with the scheduler service rate as discussed above. Moreover, all scheduling algorithms converge to an average response time of 2.2 seconds at service rates greater than 100 requests/second. Even then, limiting session queues improves performance as compared to without limiting session queues.
- When the online rate-setting algorithm is used along with the LSF scheduling policy, we obtain similar performance improvements at approximately 0.5 seconds. The average service rate set by online rate setting was 17 requests/second when $\alpha = 0.3$ and the rate is updated every 10 seconds.

C.2 Asymmetric Attack

DDoS-Shield improves the performance under the most-potent asymmetric attack from 10 seconds to 0.8 seconds, as seen from Figure 12(b). Note that under asymmetric attacks, the performance of DDoS-Shield is much more sensitive to admittance of attack sessions. At service rates higher than 17 requests/sec, the response times increase sharply for even the best scheduling algorithms (LSF, PSS), with their performance becoming similar to that of the baseline schedulers. The reason is that at high service rates, a slight increase in admittance of attack sessions drives the server CPU loads to as high as 100%, as depicted in Figure 13.

C.3 Repeated One-shot Attack

Similarly, for repeated one-shot attacks, DDoS-Shield improves the performance under the most-potent attack (inter-session time=0 seconds) from 40 seconds to 1.5 seconds. The

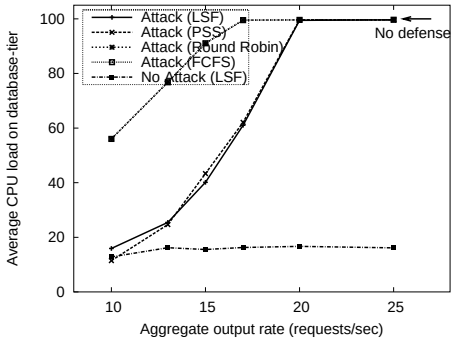


Fig. 13. Database CPU load under asymmetric attack

best performance is achieved using LSF scheduler at scheduler service rates of approximately 15 requests/second.

C.4 Moderate Potency Attacker Strategies

Recall from our discussions in Section IV that lower potency attacks are more difficult to detect than high potency attacks. Hence, to demonstrate the efficacy of DDoS-Shield in thwarting moderate potency attacks, we evaluate the performance under varying request flooding and varying asymmetric attack strategies. Using the scheduling policy LSF and the service rate set at 15 requests/second, DDoS-Shield maintains the performance of normal sessions at 0.8 seconds, even when the attack rate was varied by changing the think-time over $[0 - 7]$ seconds. Similarly, DDoS-Shield maintains the performance of normal sessions at 0.5 seconds, even when 300 attack sessions morph their workload profile by employing lighter requests alongside the heavy BestSellers requests. Finally, DDoS-Shield maintains performance at 1.5 seconds, even when the repeated one-shot attack is varied by changing the attacker inter-session times.

The success of DDoS-Shield in thwarting moderate potency attacks as well as high potency ones, is due to the suspicion assignment mechanism being able to differentiate between legitimate and malicious sessions rapidly. Recall from Figure 10(a) that even though lower-rate or lower-intensity attacks are detected with certitude much later than their high potency counterparts, on average they are assigned higher suspicion than normal sessions after only 4 requests. Hence, they are quickly given lower priority service by the LSF scheduler compared to the legitimate sessions.

VI. Related Work

CERT[3] classifies denial of service attacks in three broad categories: 1) attacks aimed at consumption of scarce resources such as network bandwidth or CPU; 2) attacks aimed at destruction or alteration of configuration information; and 3) attacks aimed at physical destruction or alteration of network components. This paper focuses on a class of attacks in the first category, namely attacks mounted at the application layer (layer-7) with attackers posing as legitimate clients of the service. The attack classes we consider overwhelm server resources in the web cluster and hence are distinct from earlier attacks that have primarily targeted network connectivity. Most recent examples of network attacks mimicked flash crowds using zombie clients [11][15].

[3]http://www.cert.org

A. Detecting DDoS attacks

The first step in thwarting a DDoS attack is to detect it. Existing detection mechanisms operate at the network level to detect DDoS floods in the network [4], [17], [18], [23]. For example, the anomaly detection system in [17] assigns every packet a score based on the probability of it being a legitimate packet given the attribute values it carries. The attacks we are focusing in this paper cannot be detected by these tools as they do not necessarily flood the network with high volumes of traffic.

Other detection mechanisms attempt to catch intrusions both at the network and the host level [24]. While the attacks in this paper do not rely on intrusions at the victim, effective intrusion detection makes it difficult for the attackers to commandeer client machines, and hence could only act as a first-step defense with reference to our attacks.

Distinguishing a DDoS attack from a flash crowd has also proven difficult. Two properties to make the distinction are identified in [13]: (1) a DoS event is due to an increase in the request rates for a small group of clients while flash crowds are due to increase in the number of clients; and (2) DoS clients originate from new client clusters[4] as compared to flash crowd clients which originate from clusters that had been seen before the flash event. These characteristics may not help distinguish the attacks discussed in this paper since (1) it is difficult to associate the amount of resources consumed to a client machine and (2) botnets consisting of geographically wide-spread machines are increasingly likely to belong to known client clusters. In contrast, our suspicion assignment mechanism observes the *behavior* of the clients to detect suspicious activity.

Our suspicion assignment mechanism relies on statistical methods. However, our problem formulation differs from similar techniques, such as sequential hypothesis testing [14][26] in two respects: First, we define only one hypothesis for legitimate behavior, and the hypothesis for malicious behavior is interpreted as anything which does not follow the legitimate hypothesis. Thus, not relying on an alternate hypothesis for the attackers gives our scheme the ability to detect misbehaviors not seen yet. Second, unlike sequential tests which output binary decisions of legitimate or malicious while bounding detection and false-positive probabilities, we output a continuous measure of suspicion. This gives our scheme the ability to start penalizing misbehaving sessions as soon as their suspicion becomes distinct from that of legitimate sessions.

B. Counter-DDoS Mechanisms

In [15], Kandula et al. design a system to protect a web cluster from DDoS attacks by (1) designing a probabilistic authentication mechanism using CAPTCHAs (acronym for "Completely Automated Public Turing test to tell Computers and Humans Apart") and (2) designing a framework that optimally divides the time spent in authenticating new clients and serving authenticated clients. Unfortunately, requiring all users to solve graph puzzles has the possibility of annoying users and introducing additional service delays for legitimate users. This also has the effect of denying web crawlers access to the site and as a result search engines may not be able to index the content. Finally, new techniques may render the graphical puzzles solvable using

[4]A Client cluster is defined as a group of topologically close clients, identified via BGP routing tables.

automated methods [20]. The DDoS-Shield does not depend on Turing tests; instead, it uses statistical methods to detect attackers and employs rate-limiting through request scheduling as the primary defense mechanism.

The technique of rate-limiting unwanted or hostile traffic has often been used as a counter-measure against DDoS attacks. For example, network packets deemed suspicious could be dropped [17] or rate-limited [12]. The class-based queuing scheme used in [16] uses a load balancer to block or limit service to client IP addresses depending on their bandwidth consumption patterns. Similarly, the probabilistic queuing scheme in [25] uses a randomized LRU cache to regulate bandwidth consumption to malicious clients. At the infrastructure level, schemes for routers to cooperatively block malicious traffic were proposed in [5]. Such techniques are all geared towards countering high bandwidth flows reminiscent of today's DDoS attacks. In contrast, by rate limiting the work a server cluster performs, we can prevent attacks on both network bandwidth as well as those that are aimed at other types of system resources, such as CPU or storage.

VII. Conclusions

In this paper, we explored the vulnerability of systems to sophisticated layer-7 DDoS-attacks which are both protocol-compliant as well as non-intrusive. These attacks mimic legitimate clients and overwhelm the system resources, thereby substantially delaying or denying service to the legitimate clients. We developed a framework to classify these resource attacks as one of request flooding, asymmetric workload, repeated one-shot attacks or combinations there-of, on the basis of the application workload parameters that they exploit. Since these resource attacks are un-detectable via sub-layer-7 techniques, we developed DDoS-Shield, a counter-mechanism which assigns a suspicion measure to a session in proportion to its deviation from legitimate behavior and uses a DDoS-resilient scheduler to decide whether and when the session is serviced. Using a web application hosted on an experimental testbed, we demonstrated the potency of these attacks as well as the efficacy of DDoS-Shield in mitigating their performance impact.

References

[1] TPC-W: Transaction Processing Council . http://www.tpc.org.

[2] C. Amza, E. Cecchet, A. Chanda, A. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel. Bottleneck characterization of dynamic web site benchmarks. Technical Report TR-02-391, Rice University, February 2002.

[3] C. Amza, A. Cox, and W. Zwaenepoel. Conflict-aware scheduling for dynamic content applications. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS'03)*, Seattle,WA, March 2003.

[4] Service provider infrastructure security: detecting, tracing, and mitigating network-wide anomalies. http://www.arbornetworks.com, 2005.

[5] K. Argyraki and D.R. Cheriton. Active internet traffic filtering: Real-time response to denial-of-service attacks. In *Proc. of USENIX Annual Technical Conference*, April 2005.

[6] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel. Scalable content-aware request distribution in cluster-based network servers. In *Proceedings of the USENIX 2000 Annual Technical Conference*, June 2000.

[7] N. Bhatti, A. Bouch, and A. Kuchinsky. Integrating user-perceived quality into web server design. In *Proceedings of the 9th International World Wide Web Conference*, Amsterdam, Netherlands, May 2000.

[8] T. Calinski and J. Harabasz. A dendrite method for cluster analysis. *Communications in Statistics*, 3:1–27, 1974.

[9] President's Information Technology Advisory Committee. Cyber security: A crisis of prioritization. www.hpcc.gov/pitac/reports/20050301_cybersecurity/cybersecurity.pdf.

[10] Cover and Thomas. *"Elements of Information Theory"*. Wiley, 1991.

[11] California Central District. United states vs jay echouafni et al. (operation cyberslam). www.usdoj.gov/criminal/fraud/websnare.pdf.

[12] A. Garg and A. L. N. Reddy. Mitigating denial of service attacks using qos regulation. In *Proceedings of International Workshop on Quality of Service (IWQoS)*, 2002.

[13] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites. In *Proceedings of the International World Wide Web Conference*, pages 252–262. IEEE, May 2002.

[14] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *Proceedings of IEEE Symposium on Security and Privacy*, Oakland, May 2004.

[15] S. Kandula, D. Katabi, M. Jacob, and A. W. Berger. Botz-4-sale: Surviving organized ddos attacks that mimic flash crowds. In *Proceedings of Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, May 2005.

[16] F. Kargl, J. Maier, and M. Weber. Protecting web servers from distributed denial of service attacks. In *World Wide Web*, pages 514–524, 2001.

[17] Y. Kim, W. C. Lau, M. C. Chuah, and H. J. Chao. Packetscore: Statistics-based overload control against distributed denial-of-service attacks. In *Proceedings of Infocom*, HongKong, 2004.

[18] Mazu profiler. http://www.mazunetworks.com, 2005.

[19] G. W. Milligan and M. C. Cooper. An examination of procedures for determining the number of clusters in a data set. *Pyschometrika*, 50:159–179, 1985.

[20] G. Mori and J. Malik. Recognizing objects in adversarial clutter: Breaking a visual captcha. *IEEE Computer Vision and Pattern Recognition*, 2003.

[21] The Honeynet Project and Research Alliance. Know your enemy: Tracking botnets. http://www.honeynet.org.

[22] S. Ranjan, R. Karrer, and E. Knightly. Wide area redirection of dynamic content in internet data centers. In *Proceedings of IEEE INFOCOM*, Hong Kong, 2004.

[23] L. Ricciulli, P. Lincoln, and P. Kakkar. TCP SYN flooding defense. In *Proceedings of CNDS*, 1999.

[24] Tripwire enterprise. http://www.tripwire.com, 2005.

[25] S. Voorhies, H. Lee, and A. Klappenecker. A probabilistic defense mechanism against distributed denial of service attacks.

[26] A. Wald. *Sequential Analysis*. J. Wiley and sons, New York, 1947.