RICE UNIVERSITY

High Performance Distributed Denial-of-Service Resilient Web Cluster Architecture

by

Supranamaya Ranjan

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Doctor of Philosophy

APPROVED, THESIS COMMITTEE:

[illegible]

Dr. Edward W. Knightly
Associate Professor, Chair
Electrical and Computer Engineering

[illegible]

Dr. Peter Druschel
Professor
Computer Science

[illegible]

Dr. T. S. Eugene Ng
Assistant Professor
Computer Science

Houston, Texas

October, 2005

UMI Number: 3216765

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI®

UMI Microform 3216765

Copyright 2006 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

High Performance Distributed Denial-of-Service Resilient Web Cluster Architecture

Supranamaya Ranjan

Abstract

Though the WWW has come a long way since when it was monikered the World Wide Wait, it is still not reliable during heavy workload conditions. Overloads due to flash arrival of users or diurnal workload patterns are known to exponentially increase download times. More recently, online banks and portals have been the target of Distributed Denial-of-Service (DDoS) attacks, which send a deluge of requests and drive away the legitimate users. This dissertation proposes a web hosting architecture consisting of a grid of clusters, to provide high-performance in the presence of standard overload conditions as well as resilience during attacks.

The architecture's high-performance component is provided by a server selection framework, *W*ide-*A*rea *ReD*irection (WARD), which efficiently multiplexes resources across the cluster grid. Traditional approaches assume that minimizing network hop count minimizes client latency. In contrast, WARD's server selection algorithm forwards requests to the server that minimizes the total of estimated network and server delays. WARD is better-suited to handling overload conditions in dynamic web content, which are known to stress compute resources more than the network. Using a combination of analytical modeling and testbed experiments, it's shown that delay savings by redirecting requests to an under-loaded cluster can far outweigh the overhead in inter-cluster network latency. For instance, for an e-commerce site with 300 concurrent clients, redirection reduces download times from 5 to 2.3 seconds.

The architecture's DDoS-resilience is provided by DDoS-Shield, consisting of a suspicion assignment mechanism and a scheduler. Assuming sophisticated attackers, the possible attacks are characterized as either request flooding, asymmetric or repeated one-shot, on the basis of the application workload parameters exploited. In contrast to prior work, the suspicion mechanism assigns a continuous valued vs. binary suspicion measure to each client session, and the scheduler utilizes these values to determine if and when to schedule a session's requests. Testbed-driven experiments demonstrate the potency of these resource attacks as well as evaluate the efficacy of the counter-mechanism. For instance, an asymmetric attack effected to overwhelm the database CPU, increases download times from 0.15 to 10 seconds, while DDoS-Shield is shown to improve performance to 0.8 seconds.

Acknowledgments

"Every search begins with the beginner's luck, and ends with the victor's being severely tested."

These lines from *The Alchemist* [1], succinctly capture the five and a half year journey that this dissertation has been. While fate and my quest for knowledge conspired together to bring me to Rice University in August 2000, the pursuit was defined completely much later under the expert mentorship of my advisor, Professor Edward Knightly. I would like to thank Prof Knightly, for providing me with this great opportunity and for building the right environment at Rice Networks Group (RNG) for independent thinking and quality research. Our interactions shaped my inchoate thoughts into the polished form that is this dissertation. Our weekly discussions were crucial in keeping this dissertation on the right track and would bring me out of the blind alleys I would often turn into. The intellectual aspect to our interactions was only one side to the story, I have learnt so much more from him about things as varied as efficiency, diligence and professionalism to name a few. But above all, thanks for believing in me all this while.

I am thankful to my dissertation committee members, Professor Peter Druschel and Professor Eugene Ng for their help and guidance. Constant communication with them regarding the status of my dissertation was a great exercise in making my thoughts coherent, while their suggestions gave this dissertation the sheen that it has.

This dissertation encompasses collaborations and friendships built with some great researchers. I am grateful to Roger Karrer for the insightful discussions we had during the spring of 2003, which led to the germination and eventual fruition of our work on wide-area redirection and above all for the fun time we had while presenting this work at Infocom in Hong Kong. I am indebted to Cristiana Amza, Alan Cox, Willy Zwaenepoel and several other researchers from the Rice CS Systems Group, for their feedback on wide-area redirection as well as providing me with the initial TPC-W implementation upon which I built and tested wide-area redirection and DDoS-Shield. I am deeply grateful to Mustafa Uysal and Ram Swaminathan for the intense rounds of discussions we had during the spring of 2005, leading to the development of ideas mentioned in DDoS-Shield. Ram's Socratic style of discussions were monumental in clarifying my thoughts on anomaly detection. In him I also found a mentor with whom I could discuss various issues related to a career in research, thanks Ram.

Weekly meetings with RNG provided me with a great forum to present ideas and gain important feedback. The RNG was the toughest audience to the presentations I made related to this dissertation and their critical eye for detail helped me hone my presentation skills tremendously. Above all, I gained wonderful friends, whom words are not enough to thank: Aleksander, for the great time we had during the summer of 2003 stealing grapes (while discussing research ofcourse!) in Napa Valley; Violeta, for her great friendship during the tough times of 2005, we survived; Joseph, for being a great office-mate and for that infectious enthusiasm; Theodoros, for being a reliable friend and the numerous discussions over beer about life and such; Dee, for her help with all things administrative and the life-saving provisions of espresso and food; and all the other group members, Abdul, Ashu, Bahar, Farbod, Jingpu, Joshua, Omer, Ping and Rahul for their constant support and encouragement.

My time in Houston was made ever memorable by the wonderful friends I made during my stay here, without whom the smooth transition into the American life wouldn't have been what it was: C. Vikram, Mani and Robert for the blast we managed to have despite the graduate workload at Rice; Aditya, Auleen, Don, Feby, Gullu, Ingar, Kanodia, Kiran, Nitin, Rajesh, Rajeshwari, Shreya and Shubha for being the best friends ever; Talwar uncle and Rita aunty for being the guardian angels and my spiritual gurus; and the dearest, my Viqulia, for her love and for being ever understanding, with whom I shared all the joys and sorrows that accompanied this dissertation, without whom all the celebrations wouldn't have been worth it and moments of dejections would have been unbearable.

I dedicate this dissertation to my family which was more overjoyed than me at its completion. My mother, also my first teacher, who doesn't know anything technical about computers or networks, would probably be the only person to read this dissertation cover to cover. My father, himself a PhD in Physics, the pride in his eyes is my reward. The happiness of badi ma, bade pa and my cousins Rakesh, Ravi and Shashi, I will always remember. And my ever loving sister, Sunny, the cheerful glee of her voice on hearing the news, I will treasure forever.

- Om -

Contents

Illustrations

Chapter 1

Introduction

1.1 Motivation

The World Wide Web (WWW) has come of age since its inception, when the first web page was served from the first web server at CERN in the early 1990's. The WWW denizens of today increasingly rely upon it for obtaining and publishing information, conducting commerce, interacting socially, and seeking entertainment. This increasing reliance on the WWW as a ubiquitous medium becomes ever more apparent whenever there is a disruption in the availability of a certain web service. Furthermore, due to the much higher access network bandwidths today than a decade ago, clients of web services have much higher expectations with the service quality and hence are less tolerant to degradation in throughput or access times.

The disruptions and degradations in a web service can be accounted for by one of the following three overload conditions: (1) *Time-of-Day* effect which is the diurnal variation in traffic observed at most web sites; (2) *Flash Crowd* arrivals which is a sudden surge in legitimate users of a web service and; (3) *Distributed Denial-of-Service (DDoS) attacks* which are deliberate attempts to overload a web service. Thus, this dissertation proposes a novel web cluster architecture to host applications which optimizes the client access times inspite of the presence of either of these overload traffic conditions. The architecture is a global-scale grid of clusters, each cluster hosting the same content and inter-connected to other clusters through custom high-bandwidth links. A majority of the contemporary e-commerce companies host their services on similar architectures. For instance, according to a recent (January 2005) interview on CBS's 60 minutes, the search-engine giant Google is known to use 100,000 servers spread across 60 clusters. Optimizing the client access times for the web applications for all kinds of traffic is challenging owing to several reasons: distributed nature of the applications, inaccuracies in predicting traffic-load and classifying traffic as legitimate or malicious.

A few observations motivate this dissertation greatly. First, the need for an efficient end-system is highlighted by the observation that the bottlenecks in accessing a web service are shifting away from the network to the compute resources of the service. There is far greater bandwidth today at the core of the Internet as well as network access points than two decades ago. Moreover, most of the bandwidth in the Internet core is over-provisioned [2, 3] due to which delays across network backbones are increasingly dominated by speed-of-light delays, with minimal router queuing delays.

Moreover, the increasing complexity, scale and size of offered web services adds towards the higher resource consumption and hence bottlenecks at the end-systems. For instance, one of the important brand differentiation strategies adopted by e-commerce sites is personalizing content to a user's preferences or locality. Serving personalized content often requires generating the content dynamically using PHP, JSP, CGI or other application server methods, after taking the user's preferences, past history and geographic location into account. Common examples of personalized content include generating a list of top-selling books or movies in the genre of preference of a certain user. Another important use of generating content dynamically is the ability to embed real-time data e.g., generation of latest stock quotes at brokerage sites or the generation of latest auction price at e-auction sites. Generating dynamic pages requires interfacing with a database server and obtaining the relevant data in real-time i.e., as and when the page is requested by a user. In contrast, static content which change rarely during consecutive visits to the same page, such as images and text are served directly from the web server, without the need for any interfacing with the database. Most of the current cluster architectures are engineered towards serving static content efficiently, however, since dynamic content generation is much more compute intensive in contrast with static content, there is a pressing need for a rethink in the design of a cluster architecture geared towards dynamic content. Thus, the scope of the architecture introduced here is restricted to dynamic content web-sites.

Specifically, the dissertation minimizes client download times during overload conditions by considering them as either non-attack outage conditions (standard time-of-day variations or flash crowds) or, attack outage conditions (DDoS attacks) and developing a framework to handle each of them. First, the non-attack outage conditions are handled by viewing the grid of clusters as a global resource pool and efficiently utilizing the resources in this pool. This is done by a server selection protocol namely *W*ide *A*rea *R*edirection of *D*ynamic Content (WARD), which redirects requests to the best server, which could be either in the local cluster or one of the clusters remote to the user. Thus, if a certain cluster is overloaded either due to the sudden popularity of the application in the corresponding geographic region or due to the fact that it is day time in this region, then requests can be redirected away to potentially under-loaded remote clusters. Second, this dissertation proposes a mechanism, namely DDoS-Shield to limit the impact of DDoS attacks which overload cluster resources. The counter-mechanism consists of anomaly detection techniques which assign a measure of suspicion to each client session. However, we can only detect attack sessions with absolute certitude after observing them for a long time, by which time the damage is done. Hence, the counter-mechanism also consists of a scheduling framework in which sessions are monitored from inception, and their service rates reduced as and when their suspicion values increase. Moreover, to limit the impact due to false-positives (legitimate sessions interpreted as malicious), the scheduler never drops a session whenever there is adequate capacity in the system.

The rest of this chapter discusses the contributions made through this dissertation in greater details. Sections 1.2 and 4.2 discuss time-of-day variations, flash crowds and DDoS attacks in the following details: the performance degradation characterized by each; the current state-of-the art techniques in handling each and; the limitations of the current techniques. Finally, the chapter is ended by Section 1.4, which summarizes the contributions made through this work.

1.2 Non-attack Outage Conditions

There are numerous incidents that highlight the performance impact of overloads due to time-of-day variations or flash crowd effects. This section discusses evidence that indicate the same and also presents the limitations of current approaches.

Several workload analysis studies [4][5] of e-commerce web sites indicate the presence of a strong diurnal variation in web traffic also known as the time-of-day effect. Web traffic has been found to vary by a factor of between 2 and 20 throughout the day, peaking during the day and troughing during the night. Current web cluster architectures are manually configured and cannot automatically adapt to these workloads. When these architectures are provisioned to handle the average workload, they suffer significant performance degradation when loads exceed capacity. In contrast, if these architectures are provisioned to handle the peak workload, they result in poor resource utilization for most of the time.

Sudden arrival of users at a site i.e., flash crowd events impact performance significantly due to the unexpected time of arrivals or magnitude of arrivals or both.

As a result, users delay each other by several orders of magnitude, sometimes even denying access completely to some of them. The inefficacy of web sites to protect from flash crowd events was made evident during the World Trade Center attacks on September 11, 2001. Millions of people tried to access information from news sites such as BBC.com and CNN.com simultaneously and either encountered delays as large as several tens of minutes or were never able to access the page requested. Similar flash crowd events have been reported when a particular web page attains popularity on being linked from news sites such as Slashdot [6], thereby leading to the alternate term "slashdot effect" for these events. One of the first known flash crowd effect, was the now famous Victoria Secret webcast in 1999, during which millions of users logged into the video stream of a fashion show. Although the arrival time of the users was known, the magnitude of arrivals wasn't. Thus, the web site which wasn't provisioned adequately to handle the workload, suffered huge delays.

Several approaches have been proposed for improving the client access times during such overload conditions at *static* content web sites. Content Distribution Networks such as Akamai [7], Digital Island [8] and Mirror Image [9] deal with the overload events under the assumption that the network is the bottleneck. Hence, they are based on the premise that minimizing the network hop-count reduces the client latencies. The primary objective of CDNs, as well as of mirroring and caching strategies, is to reduce the network latencies between the clients and the data they are accessing. This is done in two ways: Firstly, by directing clients to the closest server [10–15] and secondly, by placing the most popular *urls* on replicas closer to *hot-spots* [16, 17].

In comparison, not much progress has been made on protecting dynamic content web sites from either going down either under flash crowd arrival of users or due to the standard time-of-day patterns. One of the most common approaches is caching of dynamically generated pages so that subsequent requests towards the same page does not incur computational work at the application and database servers. Caching can occur at either the client-side with expiration times set using cookies. Alternately, dynamic pages can be cached at the front-end to the cluster such as the reverse-proxy and these pages can be expired when the database receives update queries on one of the fragments embedded in the page [18]. However, caching solutions either result in stale data being served to the clients or add to the complexity of site development and management.

Thus, this dissertation introduces a solution for hosting dynamic content web sites with the intent of protecting them from performance degradation during these non-attack overload conditions. The sites are hosted on clusters inter-connected with custom high bandwidth links thereby forming a global grid. Note that most of the popular Internet sites such as Google, Yahoo, Amazon and Microsoft are hosted on similar global grids. This dissertation proposes a novel architecture namely, *W*ide-*A*rea *R*edirection *A*rchitecture (WARD), for statistically multiplexing the infrastructure resources effectively during overload events. WARD includes an algorithm based on evidence from the test bed that server processing time for dynamic content on moderately- to heavily-loaded servers can exceed network delays by an order of magnitude. Thus, dispatchers at an overloaded cluster redirect requests away to another cluster where the request can be served quicker. WARD allows for selection of the best server i.e., one which minimizes the network plus server processing delays in accessing content.

1.3 Attack Outage Conditions

Distributed Denial of Service (DDoS) attacks pose an ever greater challenge to the Internet today with increasing sophistication of the attackers. Studies [19] estimate that DDoS attacks caused billions of dollars in revenue losses in 2003. Primarily, DDoS attacks are being used as a means to "cyber-extortion" of money from online merchants. Several such extortion attempts targeting online banks, online gaming sites and credit card processing firms have been reported in 2004. In one incident, WorldPay, a credit card processing firm was brought down for months, after their refusal to pay the "DDos Mafia" for their protection.

DDoS attackers are gaining sophistication in both the amount of resources as well as the attack methodologies adopted. Recent studies [19][20] estimate existence of farms of compromised hosts, popularly known as "botnets" as large as $60,000$ machines. Most of these machines are those which were previously compromised by a virus or worm such as CodeRed, Slammer or MS-Blast, and the attacker has set up a back-door entry into the machines through an Internet Relay Chat (IRC) channel. Usually with just a few commands through IRC, the attacker can direct these machines to start sending a flood of requests towards the victim system. A recent (May 2004) attack targeted towards the content distribution network Akamai, revealed the influence of attackers and the size of botnet used for launching the attack. In this instance, Akamai was able to trace-back the ip-address of the machine controlling the botnet, restoring access to its customer sites after approximately 90 minutes. However, future attacks may utilize a wide-array of morphing techniques, such as reducing the attack-rate or switching across different sets of botnets, thereby making attacks more difficult to distinguish. Moreover, SYN flood attacks, by large the most popular DDoS attack so far, are giving way to sophisticated application-layer (layer-7) attacks. In an attack code named CyberSlam by the FBI, an online merchant employed an alleged "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 [21].

Most attacks so far have targeted network bandwidth around Internet subsystems such as routers, Domain Name Servers, or web cluster. 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 have been found to become the bottleneck much before the network. Anticipating a future shift in the trend of DDoS attacks from network to server resources, this dissertation explores the vulnerability of Internet applications to sophisticated layer-7 attacks and develops counter-attack mechanisms.

In studying new classes of attacks, this dissertation considers 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 parameters available for the attacker to exploit are those for the application workload.

Operating under the methodology that for the best defense, one must assume the worst adversary, this dissertation assumes sophisticated layer-7 attacks which are protocol-compliant and non-intrusive. Since, the only possible way to overload a system in a protocol-compliant and non-intrusive manner is to send requests that are legitimate, these attacks must exploit certain workload parameters. For instance, an attack session may send requests at rates higher than normal or, an attack session may send higher proportion than normal of those requests which are resource-intensive. Thus, as a first step in protecting from these attacks, they are classified into several types on the basis of the workload parameters exploited.

This dissertation integrates DDoS-resilience into the proposed cluster architecture to protect the hosted site from protocol-compliant non-intrusive layer-7 attacks. DDoS-resilience comes from the following two components: First, a statistical anomaly detector based on standard as well as certain introduced statistical techniques, detects whether a session is malicious or legitimate and assigns it a suspicion probability; Second, a DDoS-scheduler uses the continuous measure of suspicion to schedule sessions into the cluster in accordance with their suspicion.

1.4 Contributions

As its main contribution, this dissertation introduces a cluster architecture for hosting dynamic content web sites which is both high-performance and DDoS-resilient. The rest of this section enlists the contributions made through this dissertation in each of the two components of the cluster architecture: Wide Area Redirection Architecture and; DDoS Shield.

1.4.1 Wide Area Redirection Architecture

This dissertation introduces WARD (*W*ide *A*rea *R*edirection of *D*ynamic content), a novel architecture for redirection of dynamic content requests from an overloaded cluster to a remote replica. The key objective is to minimize end-to-end client delays by determining at the cluster whether the *total* networking plus server processing delay is minimized by servicing the request remotely (via redirection) or locally. In particular, in this architecture, client requests are first routed to an initial cluster via any mechanism available today (e.g., from simple DNS round-robin to more sophisticated server selection schemes, as described in [14]). Upon arrival at the initial cluster, a *request dispatcher* uses a measurement-based delay-minimization algorithm to determine whether to forward the request to a remote or local server. Thus, unlike previous approaches (see Section 4.5), WARD performs cluster-driven request redirection, integrates networking and server processing delays and thereby minimizes the total delay, and requires no changes to clients, DNS or web servers.

Next, the dissertation proposes an analytical model to characterize the effects of wide area request redirection on end-to-end delay. The model consists of a system of dispatchers, M/G/1 queues that represent servers, and inter-server delays that capture the cost of redirecting to a remote cluster. With this model, we derive an expression for the optimal percentage of requests that should be dispatched to a remote cluster replica under given server and network characteristics. Moreover, we compute the expected average request response time under this dispatching policy. We then perform a systematic performance analysis to study the impact of key performance parameters such as server load, inter-cluster network latency, and measurement errors that can be expected in realistic systems.

Finally, the dissertation describes a testbed consisting of (1) clients emulating an e-commerce workload based on TPC-W benchmark [22], (2) wide area network links emulated via Nistnet [23], (3) a web server tier, (4) a request dispatcher that performs remote and local redirection via the algorithms as described above, and (5) a database tier that processes requests. With this testbed, we perform a number of experiments with example findings as follows. First, we find that that the analytical model provides a close match with experimental results for server loads up to 70%. For higher loads, effects unique to the implementation (e.g., frequent database table conflicts) lead to a deviation of predicted and measured values. Second, in spite of the differences, both model and implementation results indicate significant gains of the cluster request redirection technique. For example, for an e-commerce site with 300 concurrent clients, wide area redirection reduces the mean response time by 54%, from 5 sec to 2.3 sec.

1.4.2 DDoS Shield

As its next contribution, this dissertation proposes a framework called DDoS-Shield, to protect a generic end-system such as web clusters from DDoS attacks. First, it explores the entire range of workload parameters that an attacker can exploit to characterize layer-7 resource attacks into three classes: (1) *request flooding attacks* that send requests at rates higher than normal; (2) *asymmetric attacks* that exploit asymmetry in workload to send heavier requests towards the application; and (3) *repeated one-shot attacks*, a degenerate case of asymmetric 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. Thus, an HTTP flood can stress the network resources by organizing itself as a request flooding attack, if each session sends requests to download images at very high rates. The HTTP flood can stress the server resources as an asymmetric attack, if the attack sessions send requests involving high-computation database queries or, as a repeated one-shot attack, when each asymmetric attack session sends only one or several heavy requests per session.

As a proof-of-concept evaluation of this framework of attack classification, we evaluate server attacks on web applications. These server attacks are based on the observation that web applications, typically those that involve dynamic content, i.e., uncacheable content that is generated dynamically per client request, bottleneck on their server resources much before the network [24][25]. Furthermore, 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 in the case of HTTP floods, this dissertation is the first to demonstrate the existence of 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, server attacks would pass undetected by mechanisms that only detect network anomalies. Thus, as the next contribution, 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 this regard, our contributions are the following: First, 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. Correctness of the suspicion assignment mechanism is ensured by showing both analytically and experimentally that the suspicion values converge to either 0 or 1 as a sufficiently large number of requests per session are observed. Next, for asymmetric attacks, we introduce a set of *soundness principles*, that a metric must obey to assign suspicion values consistently across workload deviations. Finally, we present an algorithm that combines the suspicion in each of the three parameters to assign an aggregate suspicion measure to each session.

Next, we design a counter-mechanism namely, *DDoS-Shield*, that uses the suspicion assignment mechanism as an input to a DDoS-resilient scheduler designed to thwart attack sessions before they overwhelm system resources (see Figure 4.2). The DDoS-resilient scheduler combines the suspicion assigned to a session and the current system workload, to decide when and if a session is allowed to forward requests. The contributions here are the following: First, we develop scheduling policies, *Least Suspicion First (LSF)* and *Proportional to Suspicion (PSS) Share*, which account for suspicion in the scheduling decision. As a baseline for comparison, we also implement and study suspicion-agnostic policies such as per-session Round Robin and Shortest Job First (SJF) and First Come First Serve (FCFS) among all requests. Second, we 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 a database tier (see Figure 4.1). Legitimate client workload is emulated through an e-commerce benchmark [22]. Using this testbed a number of experiments are performed to characterize the potency of the attack classes as well as to evaluate the efficacy of the counter-mechanism, DDoS-Shield. The 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 scheduling policy and scheduler service rate are integral to 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 to 15 requests/second.
- Experiments indicate that 100 legitimate clients with 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 rest of this disseration is organized as follows: Chapter 2 presents the background on multi-tier architectures used for web clusters today; Chapter 3 discusses the Wide Area Redirection Architecture developed to multiplex the resources across the cluster-grid to limit the performance impact due to non-attack overload conditions; Chapter 4 discusses DDoS-Shield, the architecture component which protects legitimate clients from protocol-compliant, non-intrusive layer-7 DDoS attacks. Finally, Chapter 5 summarizes the conclusions and discusses future research directions.

Chapter 2

Background

Figure 4.1 depicts the four-tier architecture prevalent in today's IDCs. To illustrate this architecture, consider the requests of an e-commerce session. First, the *access tier* routes requests to the correct server cluster and performs basic firewall functions

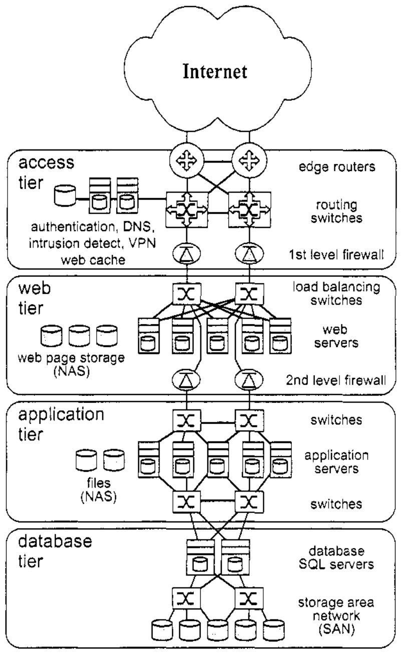


Figure 2.1 : IDC Four Tier Architecture

such as intrusion detection. Second, upon arriving at the *web tier*, load balancers may parse the request's URL and route it to a web server typically according to a load-balancing policy (e.g., using round robin or more sophisticated policies as in reference [26]). If the request is for a static web page, a server in the web tier serves the requested page. If the request is for an e-commerce functionality, it is routed to the *application tier*. The application tier orchestrates access to the *database tier* for operations such as purchase processing or maintaining the contents of the shopping cart. The application server also stores the state of the session such as the contents of the shopping cart.

Chapter 3

Wide Area Redirection Architecture

This chapter introduces WARD, (*W*ide *A*rea *R*edirection of *D*ynamic Content, a framework to efficiently multiplex resources across multiple clusters by utilizing a novel server selection algorithm. The rest of this chapter is organized as follows. Section 3.1 describes the system architecture for wide area request redirection. Section 3.2, develops a queuing model to study the architecture and Section 3.3 presents numerical studies of the fraction of requests dispatched remotely and the expected response times under varying system scenarios. Next, our testbed implementation and measurements are described in Section 3.4. Finally, Section 4.5 discusses the related work.

3.1 Redirection Architecture and Algorithm

This section describes the system model for WARD, and presents a measurement-based redirection algorithm.

3.1.1 System Model

In WARD, services and applications are replicated across all the clusters, connected using custom high-bandwidth links. Once a client request arrives at the initial cluster,[1] A dispatcher as illustrated in Figure 3.1 can potentially redirect the request

[1]Selection of the initial cluster can be static or dynamic via DNS round robin or via more sophisticated policies such as [12, 27, 28].

to a remote cluster according to the algorithm presented below. The objective of the redirection algorithm is to redirect requests only if the savings in the request's processing time at the remote cluster overwhelm the network latency incurred to traverse the backbone in both the forward and reverse path. In this way, end-to-end client delays can be reduced while requiring changes only to the dispatcher, and leaving other elements unchanged. WARD therefore provides a foundation for spatial multiplexing

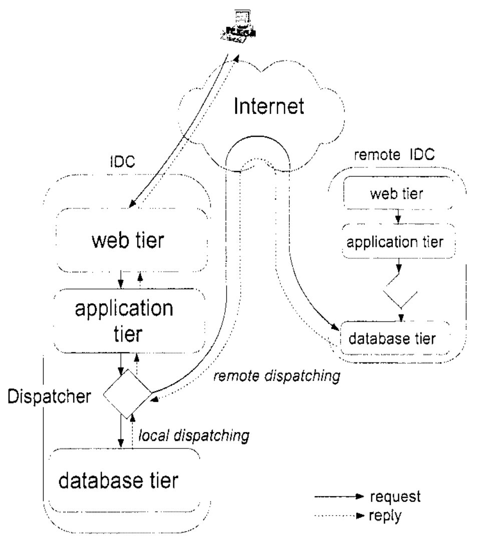


Figure 3.1 : Server Selection using WARD

of cluster resources. Namely, as a particular cluster becomes a hot-spot due to flash crowds [29, 30] or time-of-day effects [31], load can be transparently redirected to other clusters while ensuring a latency benefit to clients. For example, client access patterns have been observed to follow *time-of-day* patterns where server utilization varies with a diurnal frequency. This effect can be exploited such that no cluster has to provision for the peak demand. Thus, when the workload to one cluster is peaking, the workload at an cluster several time zones away will be much lower, enabling a significant performance improvement by allowing redirection among clusters.

3.1.2 Redirection algorithm

The objective of the redirection algorithm is to minimize the total time to service a request. Namely, if a request arrives at cluster k, then the objective is to dispatch the request to cluster j satisfying

$$\text{argmin}_j \, (2\,\Delta_{kj} + T_j) \tag{3.1}$$

where Δ_{kj} denotes the network delay between cluster k and j and T_j is the request's service time at cluster j.

In practice, the actual service time at each remote cluster T_j cannot be known in advance. Yet, it can be estimated from the average load at cluster j as well as the request type. Thus, a measurement-based algorithm is employed in which the average T_j is estimated from ρ_j, the mean load at cluster j, as well as the request type. In WARD, this is achieved by measuring a mean delay vs. load profile for each request type. clusters then periodically exchange load information to refine their estimates of each others' processing delays. In contrast, Δ_{kj} remains relatively static among clusters due to their high-speed interconnection links. Thus, on request arrival, the dispatcher uses the measured load at cluster j on the delay vs. load profile corresponding to this request's type to estimate the total service time on cluster j.

Next, let's consider a second policy which does not make a decision on a per request basis but rather computes a fraction of requests to be remotely dispatched. In particular, we show in the next section, that under certain simplifications there is an optimal ratio of requests that should be remotely dispatched in order to minimize the delay of all requests. Once this ratio is known, the dispatcher can simply remotely redirect a request with the computed probability. These two redirection policies are referred to as *per-request* redirection (or, equivalently *per-query* redirection) and *probabilistic* redirection.

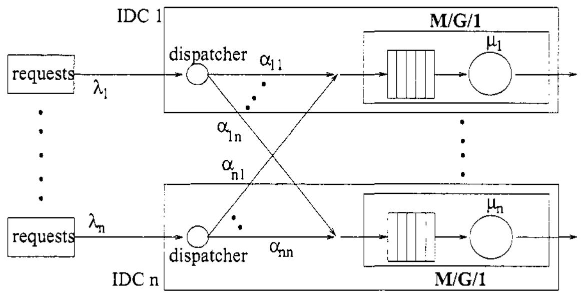


Figure 3.2 : cluster system model.

3.2 Performance Model

This section develops a performance model for wide area redirection. The methodology adopted is the following. First, for a given workload, mean and variance of service time, and network latency, an expression is derived for the delay-minimizing fraction of requests that a dispatcher should redirect to remote clusters. Next, the average total response time including service- and waiting-times and end-to-end network latency is computed. Finally, a systematic performance analysis is performed to estimate the optimal dispatching ratios α^* and to predict the expected average request response time under varying parameters, such as the server load, the end-to-end network latency and the average request service time.

Figure 3.2 illustrates the system model for WARD. We model request arrivals at cluster i as a Poisson process with rate λ_i and consider a single bottleneck tier modeled by a general service time distribution having mean $\overline{x}_i$ and variance σ_i^2.

We consider a redirection algorithm in which a request is redirected from cluster j to cluster i with probability α_{ji}, i.e., we consider probabilistic redirection. Denote $E[T_i]$ as the expected total delay for servicing a request at cluster i, and denote Δ_{ji} as the one-way end-to-end network latency for a request sent from cluster j to cluster i.

For the general case of a system of n cluster replicas, denote $\mathbf{A} = \{\alpha_{\mathbf{11}}, \ldots, \alpha_{\mathbf{ji}}, \ldots, \alpha_{\mathbf{nn}}\}$ as a matrix of request dispatching fractions, $\mathbf{E[T]} = \{\mathbf{T_1}, \ldots, \mathbf{T_n}\}$ as the vector of all total delays at a cluster bottleneck tier and $\mathbf{D} = \{\mathbf{2\Delta_{11}}, \ldots, \mathbf{\Delta_{ji}} + \mathbf{\Delta_{ij}}, \ldots, \mathbf{2\Delta_{nn}}\}$ as a matrix of round-trip times from cluster i to cluster j and back. Furthermore, denote $\mathbf{L} = \{\lambda_1, \ldots, \lambda_\mathbf{n}\}$ as a vector of request arrival rates at the cluster dispatchers, $\overline{\mathbf{X}} = \{\overline{\mathbf{x}}_\mathbf{1}, \ldots, \overline{\mathbf{x}}_\mathbf{n}\}$ as the average service time, $\mathbf{C} = \{\mathbf{c_1}, \ldots, \mathbf{c_n}\}$ as the vector of squared coefficient of variation for the service times, with $c_i = \sigma_i^2/\overline{x}_i^2$.

Lemma 1 *The mean service time for the redirection policy using a dispatching fraction* $\mathbf{A}$ *is given by:*

$$\mathbf{E[T]} = \mathbf{A} \cdot \overline{\mathbf{X}} + \frac{(\mathbf{A} \cdot \mathbf{L})\overline{\mathbf{X}}^{\mathbf{2}}(\mathbf{1} + \mathbf{C^2})}{\mathbf{2}(\mathbf{1} - (\mathbf{A} \cdot \mathbf{L})\overline{\mathbf{X}})} + \mathbf{A} \cdot \mathbf{D} \tag{3.2}$$

Proof: The total service time is composed of 3 durations: (i) the network latency of transferring the request to and from the remote cluster (ii) the queuing time at the cluster and (iii) the service time at the cluster.

For symmetry reasons, in the following equations, we attribute the "costs" to the receiving cluster i. First, we assume that the network latency between the dispatcher and a local cluster $\Delta_{ii} = 0$ and hence, network latency is incurred only by requests dispatched to a remote cluster:

$$\alpha_{ji}(\Delta_{ji} + \Delta_{ij}) \tag{3.3}$$

Second, consider the mean waiting time for a request in a cluster queue before being serviced. In general, the waiting time for for an M/G/1 queue is:

$$\frac{\rho\overline{x}(1 + c^2)}{2(1 - \rho)} \tag{3.4}$$

with $\rho = \lambda\overline{x}$.

For any cluster i, the arrival rate λ is the sum of the requests that are dispatched from all clusters j to cluster i, i.e., $\lambda_i = \sum_j \alpha_{ji}\lambda_j$. With this λ, Equation (3.4) can be rewritten for a single cluster i as:

$$\frac{(\sum_j \alpha_{ji}\lambda_j)\overline{x}_i^2(1+c_i^2)}{2(1-(\sum_j \alpha_{ji}\lambda_j)\overline{x}_i)} \tag{3.5}$$

Finally, the service time for a request at cluster i is given by

$$\alpha_{ji}\overline{x}_i \tag{3.6}$$

The addition of these 3 terms for a set of clusters yields Equation (3.2). ■

From Equation (3.2), we can compute the optimal dispatching ratios that minimize the service times over all requests. In particular, let $\mathbf{A} = \{\alpha^*_{\mathbf{11}}, \ldots, \alpha^*_{\mathbf{nn}}\}$ denote the matrix of optimal request dispatching ratios.

Proposition 1 *The optimal dispatching ratios* $\mathbf{A}^*$ *are given by:*

$$\frac{\partial}{\partial\alpha}(\mathbf{A}\cdot\overline{\mathbf{X}} + \frac{(\mathbf{A}\cdot\mathbf{L})\overline{\mathbf{X}}^2(\mathbf{1}+\mathbf{C}^2)}{\mathbf{2}(\mathbf{1}-(\mathbf{A}\cdot\mathbf{L})\overline{\mathbf{X}})} + \mathbf{A}\cdot\mathbf{D}) = \mathbf{0} \tag{3.7}$$

with $\mathbf{E}[\mathbf{T}]$ defined in Equation (3.2).

To solve Equation (3.7) for all α_{ji}, we use the following constraints to reduce the number of unknowns. First, we clearly have that $\sum_j \alpha^*_{ji} = 1$. Second, $\lambda_i \geq \lambda_j \implies \alpha^*_{ji} = 0$ i.e., when considering 2 clusters with different λ, under steady-state conditions, no requests will be dispatched from the cluster with a smaller arrival rate to the cluster with a higher arrival rate.

The optimal dispatching ratios $\mathbf{A}^*$ can be used to predict the average request service time for a system of cluster replicas.

Proposition 2 *The expected request service time under optimal dispatching ratios is given by:*

$$\mathbf{E[T^*]} = \mathbf{A^* \cdot \overline{X}} + \frac{\mathbf{(A^* \cdot L)\overline{X}^2(1 + C^2)}}{\mathbf{2(1 - (A^* \cdot L)\overline{X})}} + \mathbf{A^* \cdot D} \tag{3.8}$$

Proof: Equation (3.8) follows from Lemma 1 and by using the optimal dispatching ratios from Equation (3.7). ■

3.3 Numerical results

This section first shows that wide area redirection is effective in reducing the total access delay. Next, the key performance factors that affect the total delay are studied.

The grid is assumed to consist of 2 clusters with each replica having the same average request service time $\overline{x}$. Furthermore, we assume a symmetric network with wide area latency between the two clusters: $\Delta = \Delta_{12} = \Delta_{21}$. Finally, we set $\lambda_2 = 0$, which satisfies $\lambda_1 > \lambda_2 \implies \alpha_{21} = 0$, and denote $\lambda := \lambda_1$ and $\alpha^* := \alpha^*_{11}$ for simplicity.

The dispatching ratio is computed based on Equation (3.2):

$$E[T_1] = \alpha\overline{x} + \frac{\alpha\lambda\overline{x}^2(1 + c^2)}{2(1 - \alpha\lambda\overline{x})} \tag{3.9}$$

$$E[T_2] = (1 - \alpha)\overline{x} + \frac{(1 - \alpha)\lambda\overline{x}^2(1 + c^2)}{2(1 - (1 - \alpha)\lambda\overline{x})} + 2(1 - \alpha)\Delta \tag{3.10}$$

Equations (3.9) and (3.10) are solved according to Proposition 1 to obtain the optimal dispatching ratio α^*. Henceforth, we refer to the term $1 - \alpha^*$ as the *remote redirection* ratio, i.e. the fraction of requests dispatched remotely. Then, according to Proposition 2, the expected total delay of the cluster system is given by:

$$\begin{aligned} E[T^*] &= \alpha^*\overline{x} + \frac{\alpha^*\lambda\overline{x}^2(1 + c^2)}{2(1 - \alpha^*\lambda\overline{x})} + (1 - \alpha^*)\overline{x} \\ &+ \frac{(1 - \alpha^*)\lambda\overline{x}^2(1 + c^2)}{2(1 - (1 - \alpha^*)\lambda\overline{x})} + 2(1 - \alpha^*)\Delta \end{aligned} \tag{3.11}$$

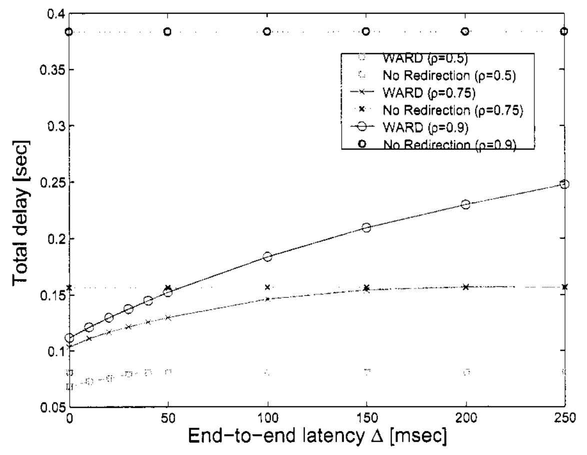


Figure 3.3 : Comparison of the total cluster delay with and without wide area redirection

If not otherwise stated, the following default values are used: $\overline{x} = 42.9$ msec, $\sigma = 40.1$ msec, where these values were obtained from the testbed and $\Delta = 36$ msec, which corresponds to a speed-of-light latency for two clusters separated by 6 time-zones at 45^o latitude. [2] we will use $\rho = \lambda\overline{x}$ to denote the total load on *all* clusters. For clusters without redirection, ρ corresponds to the server load on the bottleneck tier, whereas WARD can split this load among the local and remote clusters. To obtain a given value of ρ, the arrival rate λ will be scaled, with $\overline{x}$ remaining fixed.

3.3.1 The case for wide area redirection

First, we provide evidence that wide area redirection is able to decrease the user-perceived total delay. We calculate the total delay of WARD using Equations (3.7) and (3.8) and compare it to the total delay of a cluster without redirection. Figure 3.3

[2]Given the circumference of earth at 45^o latitude as 28335 km and the speed-of-light through optical-fiber as 205 km/sec, the one-way latency across 1 time-zone can be calculated as: $28335/(205* 24) \approx 6$ msec.

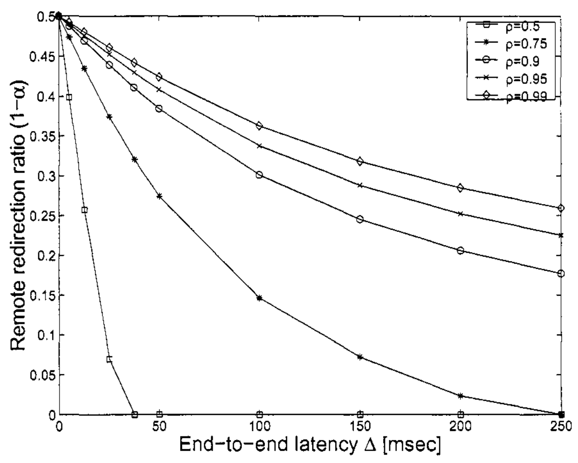


Figure 3.4 : Remote redirection ratio $1 - \alpha$ for different end-to-end latencies Δ and server loads ρ.

shows the total delay as a function of the end-to-end latency and different system loads ρ.

For a load $\rho = 0.5$, improvements are achieved only when the end-to-end latency $\Delta \leq 25$ msec. For $\Delta > 25$ msec, the redirection cost exceeds the processing time so that all requests are serviced locally. However, a significant improvement is achievable for higher loads. For a moderate load of $\rho = 0.75$ and $\Delta < 50$ msec, the total delay is reduced from 0.16 sec to <0.13 sec using WARD, an improvement of > 18%. For a heavily loaded system with $\rho = 0.9$ and when $\Delta < 50$ msec, the total delay is reduced from 0.38 sec without redirection to <0.15 sec using WARD, an improvement of > 60%. Moreover, for loads $\rho > 0.9$, still higher improvements are predicted by the model.

3.3.2 Server load versus network latency

Next, we study the influence of server load and the network latency on the redirection decision. An increased network latency implies a higher overhead to send a request to a remote cluster and here we quantify the network latencies for a given server load, until which gains can be expected out of redirection. The influence of the latency Δ for different server loads ρ on the optimal redirection ratio (Equation (3.7)) is shown in Figure 3.4. Each curve depicts a value of server load ρ and the x-axis denotes the network latency Δ between two clusters.

Results with a latency $\Delta = 0$ therefore correspond to a cluster with 2 local servers. Observe that in this case, the remote redirection ratio $1 - \alpha$, is 0.5 independent of the server load. Since no latency costs are incurred, the optimal strategy is to equally balance the load on the two local servers. We make two further observations: Firstly, for a given network latency, the model redirects a greater number of requests as the server load increases. Secondly, for a given server load, the redirection ratio decreases as the network latency is increased and goes to zero, when the cost of network latency exceeds the gain in server processing time due to redirection. For a non-heavily loaded system ($\rho = 0.5$), it is of advantage to redirect only when the network latency $\Delta < 25$ msec, while for a moderate load of $\rho = 0.75$, redirection is of an advantage for latency as high as 150 msec. For $\rho \geq 0.9$, the model predicts redirection ratios of 0.2 for latency as high as 250 msec.

3.3.3 Service time

Figure 3.5 illustrates the effect of the request service time mean $\overline{x}$ on the remote redirection ratio. The x-axis denotes the server load ρ and each curve denotes a different mean service time. For a small service time $\overline{x} = 10$ msec, the remote redirection ratio $1 - \alpha$ is 0 for server loads up to $\rho = 0.7$. Only above this high load does the dispatcher send requests to a remote cluster because the redirection costs exceed the service time on the local cluster.

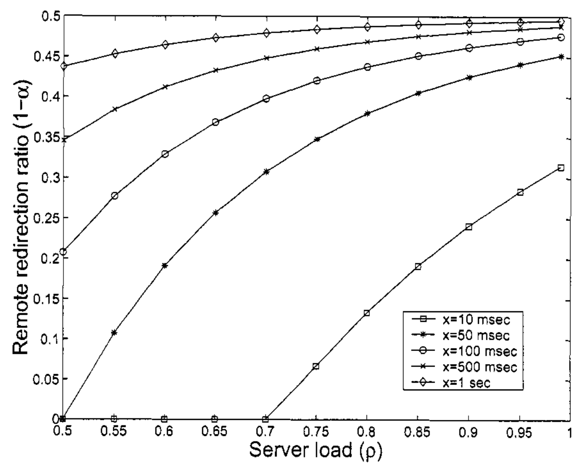


Figure 3.5 : Remote redirection ratio $1 - \alpha$ for different service times $\overline{x}$ and server loads ρ.

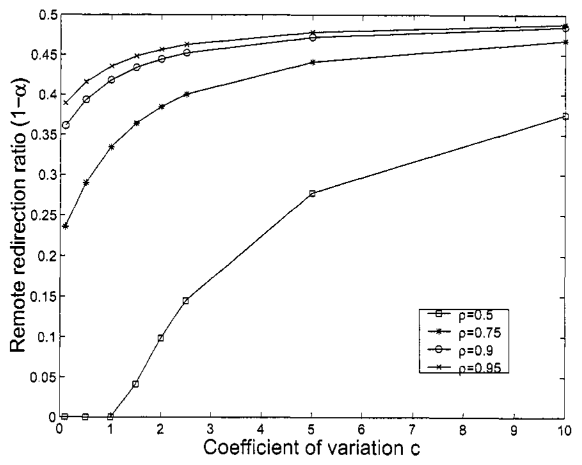


Figure 3.6 : Remote redirection ratio $1 - \alpha$ for different coefficient of variations c and server loads ρ.

When the request service time $\overline{x}$ increases, the remote redirection ratio increases for a given server load ρ. For service times of 50 msec, which is close to the testbed value, an increase in the redirection ratio starts at $\rho = 0.5$. Finally, for $\overline{x} = 1$ sec, the ratio stays above 42%.

Figure 3.6 shows the effect of the coefficient of variation c on the remote redirection ratio. It can be expected that an increase in variance leads to an increase in the redirection ratio. Figure 3.6 shows the greatest influence of the variance when c increases from < 1 to 1 for $\rho \geq 0.75$.

Hence, the model predicts an increase in remote redirection ratios when either the service time or the coefficient of variation increases. This allows us to predict that future e-commerce sites designed with greater complexity in their dynamic content would achieve still higher performance gains on using wide area redirection.

3.3.4 Measurement errors

The dispatcher bases its redirection decision on 2 measured values: the network latency Δ and the server load ρ. So far, we have assumed that perfect information is available for this decision. In this section, we study the impact of measurement errors on the effectiveness of the algorithm and quantify it in terms of *error tolerance* defined as the percentage error $\pm\epsilon$ that increases the total delay by at most 2%.

First, we study the impact of network latency errors as follows. Let Δ denote the true inter-cluster network latency and $\widehat{\Delta} = \Delta + \delta$ the measured value, and $\widehat{\mathbf{D}}$ the corresponding round-trip time matrix. The dispatcher calculates the dispatching ratios using $\widehat{D}$ (Equation (3.2))

$$\mathbf{E[T]} = \mathbf{A} \cdot \overline{\mathbf{X}} + \frac{(\mathbf{A} \cdot \mathbf{L})\overline{\mathbf{X}}^2(\mathbf{1} + \mathbf{C}^2)}{\mathbf{1} - (\mathbf{A} \cdot \mathbf{L})\overline{\mathbf{X}}} + \mathbf{A} \cdot \widehat{\mathbf{D}} \tag{3.12}$$

For the calculation of average total delay, Equation (3.8) is used with the true latency values $\mathbf{D}$. The effects of measurement error in network latency on the remote redirection ratio $(1 - \alpha)$ and the resulting average request response time are shown in Figures 3.7 and 3.8 respectively. Each curve denotes a different (true) latency Δ, and the x-axis denotes the error δ, in percent of Δ. A value of 0 on the x-axis corresponds

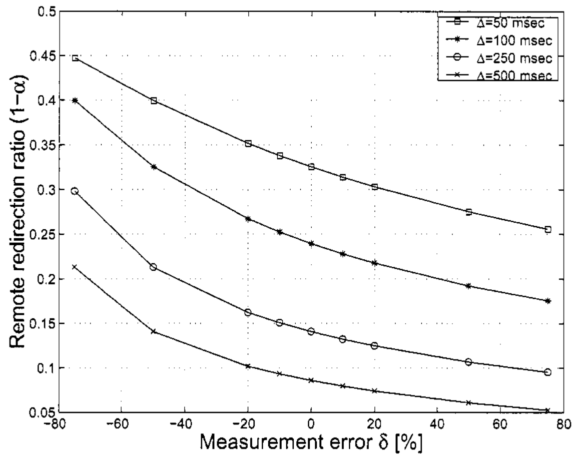


Figure 3.7 : Remote redirection ratio $1 - \alpha$ for different network measurement errors δ.

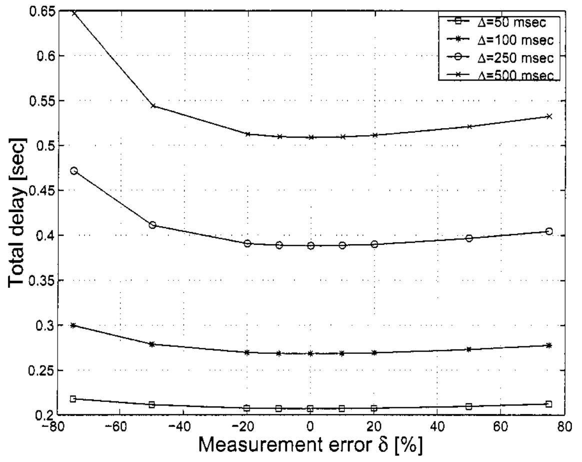


Figure 3.8 : Total delay for different network measurement errors δ.

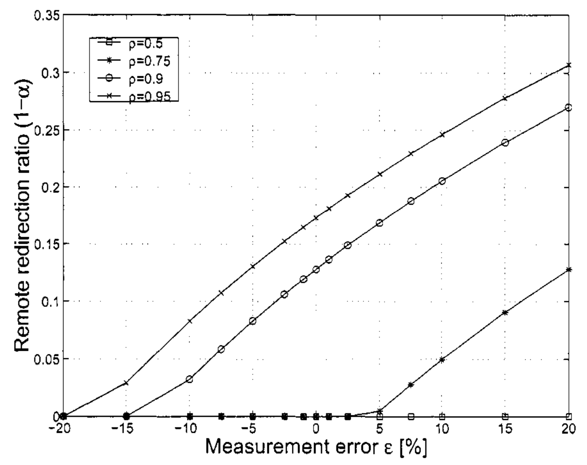


Figure 3.9 : Remote redirection ratio $1 - \alpha$ for different server load measurement errors ϵ.

to perfect end-to-end latency information. The server load is set to $\rho = 0.95$.

Figure 3.7 shows that the redirection ratio changes more for negative δ than for the corresponding positive δ. The reason is that the redirection ratio does not grow linearly with the end-to-end latency, as shown in Figure 3.4. As a consequence of the asymmetry, the total delay increases more for negative δ, as shown in Figure 3.8. Note, however, that the response times are not highly sensitive to latency measurement errors and the the error tolerance is quite high at $\pm 20\%$.

Likewise, we consider a scenario when the dispatcher has inaccurate server load measurements, e.g., due to delays in receiving the measurements. In this scenario, the measured load at the dispatcher is given by $\widehat{\rho} = \widehat{\lambda}\overline{x}$, with $\widehat{\lambda} = \lambda + \epsilon$ (where ϵ is in percent of the correct load ρ) and the corresponding measured arrival rate by $\widehat{\mathbf{L}}$. The network latency is set to $\Delta = 500$ msec.

First, consider the case of measurement error $\epsilon > 0$, when the dispatcher assumes the server load to be higher than what it is and hence it redirects more requests than the optimal. Figure 3.9 shows that the remote redirection ratio increases almost

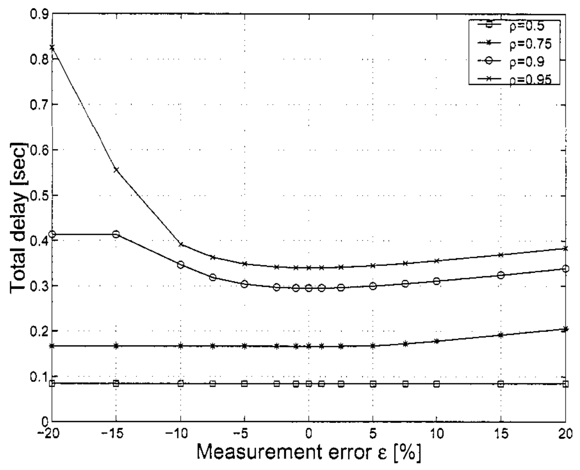


Figure 3.10 : Total delay for different server load measurement errors ϵ.

linearly for $\rho \geq 0.9$, while for $\rho = 0.75$ the ratio doesn't start increasing until $\epsilon \geq 5\%$. The extra redirections incur additional network latencies and hence the total delay also increases linearly in Figure 3.10. In particular, for $\rho \geq 0.9$, the error tolerance is $+1.5\%$. Next, consider negative ϵ, when the dispatcher assumes the local server load to be less than the actual value and hence redirects pessimistically. As a result, the load on the local server incurs greater processing times at the local cluster. As expected, Figure 3.10 shows that at high server loads $\rho \geq 0.9$, the total delay is more sensitive for negative ϵ with an error tolerance of $\approx -0.5\%$.

Thus, comparing the impact of latency and server measurement errors, the error tolerance for latency is high at $\pm 20\%$ while that for server load is an order of magnitude lower at $+1.5, -0.5\%$. Moreover, for network latency $\Delta < 500$ msec, we measure higher tolerance to errors in server load measurements. Thus, our conclusions are that (1) greater accuracy is needed in server load measurements than network latency and (2) cluster-driven redirection can have greater robustness than existing client-driven schemes as clusters can obtain accurate server load information when compared to clients, client-side proxies, or DNS servers.

3.4 Testbed implementation and experiments

This section describes the prototype implementation and testbed experiments of the WARD architecture. The results provide a proof-of-concept demonstration of wide area cluster-driven redirection, experiments into the testbed's key performance factors, and validate the performance model.

3.4.1 WARD Testbed

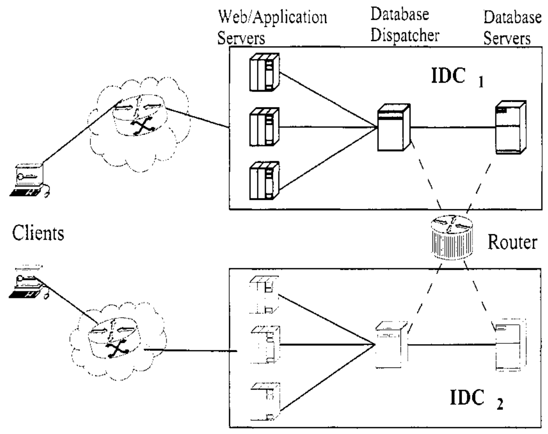


Figure 3.11 : Testbed

The testbed, depicted in Figure 3.11, consists of a cluster of Intel Pentium IV 2.0 GHz processor machines running Linux 2.4.18-14, with 512 MB SDRAM, and a 30 GB ATA-66 disk drive. One machine is configured as a router and runs Nistnet [23], an IP-layer network emulation package. Figure 3.12 shows how the router separates the machines into three domains, one for the client and two for the clusters. This setup allows variation of the network conditions (delay and bandwidth) between the client and the clusters as well as between clusters.

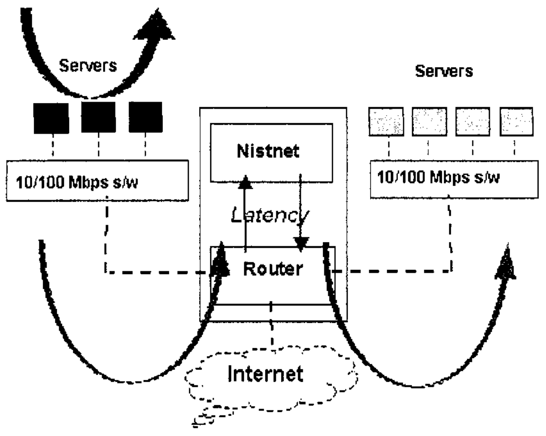


Figure 3.12 : Network Subnets with WAN delays emulated by NISTNET

Each cluster hosts a multi-tiered implementation of an e-commerce application (online-bookstore). The web tier consists of Apache web servers [32] and dynamic content is coded using PHP scripts [33] at the application tier. Access to a 4 GB database of customers and books is provided by a MySQL server [34].

Database dispatcher

A key aspect of the WARD architecture is the dispatcher which makes the decision whether to service a request at the local or a remote cluster. On this testbed, the *database* tier becomes the bottleneck first, due to the substantial processing demands of complex database queries [35]. We therefore implemented the dispatcher with remote redirection capabilities in front of the database.[3]

While the objective of the database dispatcher is to minimize the response time of the queries, its dispatching capabilities are restricted by consistency constraints. The dispatcher addresses the consistency issues by two means: maintaining an identical

[3]In the implementation, we provide the web and application tiers with sufficient resources such that the database tier is indeed the bottleneck.

total ordering of writes at all database servers, and a *read-one write-all* dispatching strategy. To maintain an identical *total ordering* of writes at all database servers, each write query is assigned a unique sequence number, and the database servers process the writes in the order of the sequence numbers. In the read-one write-all dispatching strategy, write queries are sent to all database servers, and the response is returned as soon as one of the database servers has processed the write. Hence, database consistency is maintained *asynchronously.* A read query is sent to one of the database servers where the corresponding conflicting writes have been processed using a scheduling strategy described as *conflict-aware* [35]. This *asynchronously-*maintained consistency along with *conflict-aware* scheduling has been shown to scale to higher throughputs compared to the synchronous consistency algorithms.

However, though conflict-aware scheduling limits the server set available for wide area dispatching for read queries, two factors outweigh this limitation. Firstly, read queries have a larger service time than write queries and secondly, e-commerce workloads have a significantly higher percentage of read queries [36].

WARD allows a general framework to implement remote redirection at any tier. Though we implemented it in front of the database tier, it could equivalently be done in front of the web/application tier, in which case an entire request would be redirected instead of database queries. Request redirection would incur less of network overhead, but it would constrain the queries to the local database servers, whereby the advantages due to remote redirection of queries would not be realized. We do not explore request redirection versus query redirection in this dissertation, and instead focus on the performance gains of remote redirection in general.

Redirection Algorithms

Under the consistency constraints, the database dispatcher directs read queries to the database server with the least expected response time as follows. First, from the list of clusters, all those which have unprocessed conflicting writes are excluded using conflict-aware scheduling. From the remaining clusters, the dispatcher selects an cluster by using either the *per-query* or, the *probabilistic* redirection policy. In the per-query redirection policy, the dispatcher calculates the expected response time by using measured loads of database tiers to determine if the latency overhead incurred by remote dispatching is outweighed by the savings in server processing time. In the probabilistic policy, the dispatcher uses the optimal redirection ratio computed by the model and dispatches queries with that probability. We implement the probabilistic policy such that given a number of clients, it is configured for the redirection ratio predicted by the model and hence it doesn't use the online server load measurements.

TPC-W workload

For the experimental workload, we utilize the TPC-W benchmark [22] to represent an e-commerce workload characterizing an online bookstore site. We use the implementation developed for reference [37].

The workload for TPC-W is generated by a client emulator which generates the requests specified in the browsing mix of TPC-W.[4] The client emulator opens a set of n user sessions which last 15 minutes. Each session opens a persistent HTTP connection to the web server and sends a sequence of requests to the cluster. Between two requests, the user waits for a configurable parameter termed *think time* before the next request is generated. The mean think time, which was set to 7 sec, together with

[4]Browsing mix consists of 95% read queries and 5% writes.

the number of users, defines the request arrival rate at the cluster. Note here that a PHP script can contain multiple database queries. The extraction and serialization of the queries is done by the application tier. Due to the fact that each request consists of several embedded queries, their arrival distribution and rate at the database are different from those generated by the client emulator.

3.4.2 Experimental Setup

The input parameters in the experimental study are the inter-cluster link latency which are varied through the Nistnet module and the number of clients which arrive at each cluster. The parameters measured are *request response time* as perceived by the clients, *query response time* as perceived by the database dispatcher and *remote redirection ratio* as achieved by the database dispatcher. Request response time is defined as the time elapsed between the generation of a request and the return of the last byte of the response to the client. Query response time is defined as the time period between the sending of a query by the dispatcher to the database and the reception of the response by the dispatcher. We measure the mean- and 90-%ile request (query) response time for all requests (queries) generated during the entire duration of an experiment. Remote redirection ratio is defined as the fraction of the number of queries sent by the dispatcher to a remote database server.

3.4.3 Experiments

Here, we first present the offline technique to configure the *per-query* redirection policy with the *query response time characteristics.* Second, we quantify the performance benefits of the WARD architecture by exploring the trade-off between the load on the local database server and wide area link latency. Third, we compare performance gains predicted by the analytical model of section 3.2 with those obtained via testbed

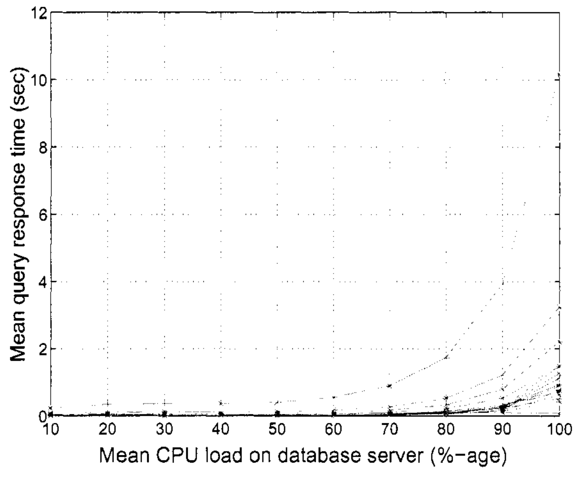


Figure 3.13 : Mean database query response times vs mean database CPU load for the 30 read-only MySQL queries in the browsing mix of the TPC-W benchmark.

measurements.

Offline measurement of query response time characteristics

In these experiments, we measure the response time as a function of CPU load, a key input to the per-query redirection policy. We use one cluster with access to one local database server. The execution time for a query depends on the number and type of other queries executing at the same time on the database server, which can be abstracted as the workload entering the system. Hence, we vary the CPU load on the database server by increasing the number of clients. In each case, we measure the mean execution time for each of the 30 read-only MySQL queries. The resulting delay-load curve as illustrated in Figure 3.13 is then used in the *per-query* redirection policy.

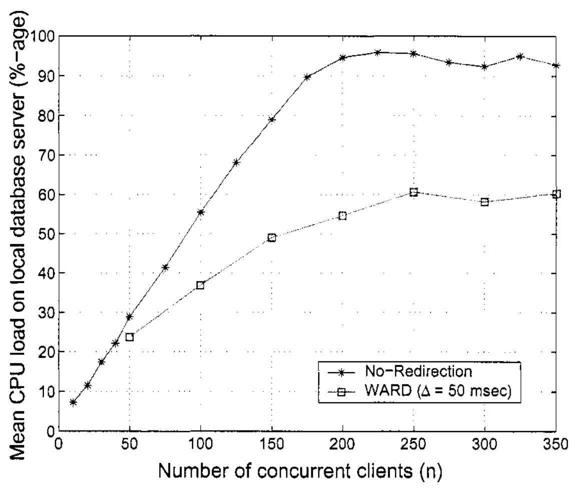


Figure 3.14 : Mean database CPU load as a function of the number of concurrent clients (n).

WARD Performance Gains

In this experiment we quantify the performance gains achieved by the *per-query* redirection policy by considering the trade-off between the two parameters of wide area link latency and CPU load on the local database server.

We compare the following two architectures: (1) *No-Redirection* architecture with two clusters, each of which has access to one local database server and doesn't employ wide area redirection and (2) *WARD* architecture with two clusters, each of which has access to one local and one remote database server and the latency between the two clusters is Δ varied as 0, 25, 50 and 100 msec. In both architectures, the workload arrives at only one cluster, termed "local," whereas the workload of the remote cluster is solely created by dispatched requests. We expect such zero-load conditions on remote clusters several time-zones away due to the *time-of-day* effects.

Figure 3.14 compares the local server CPU of a cluster without redirection to WARD with an inter-cluster latency of 50 msec. In the No-Redirection architecture, the CPU load on the database tier reaches 90% for 200 concurrent users. In contrast,

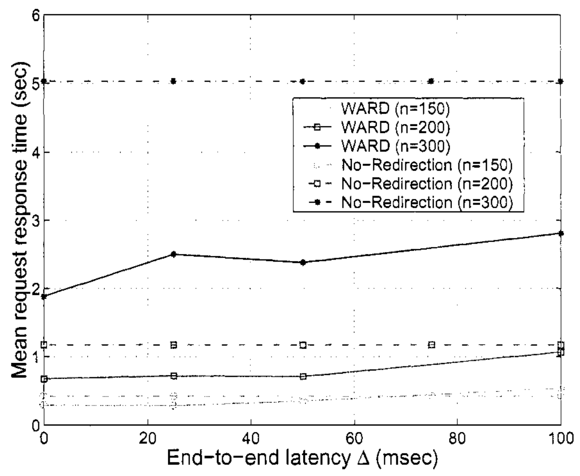


Figure 3.15 : Mean request response time as a function of Δ and the number of concurrent clients (n)

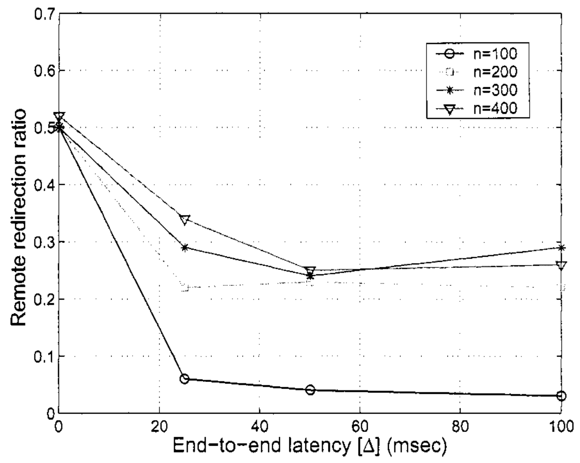


Figure 3.16 : Remote redirection ratio as a function of Δ and the number of concurrent clients (n).

WARD keeps the local database server load below 60% even for 350 concurrent users.

The high CPU load on the database server in the *No-Redirection* architecture increases the mean request response time, as shown in Figure 3.15. For 300 concurrent users, the mean request response time reaches 5 sec. In contrast, the mean request service time of WARD is 2.3 sec for an inter-cluster latency of 50 msec, a 54% re-

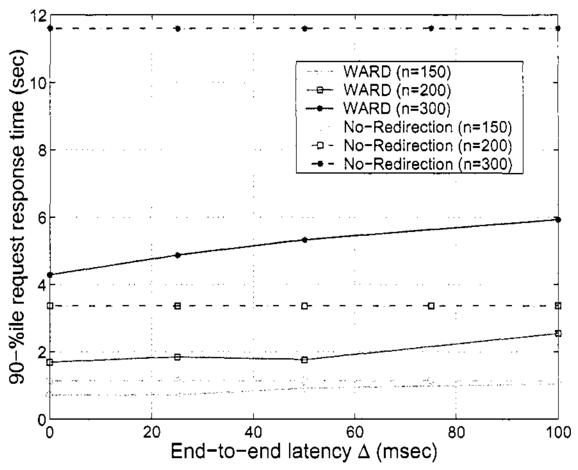


Figure 3.17 : 90 %-iles of response time as a function of Δ and the number of concurrent clients (n)

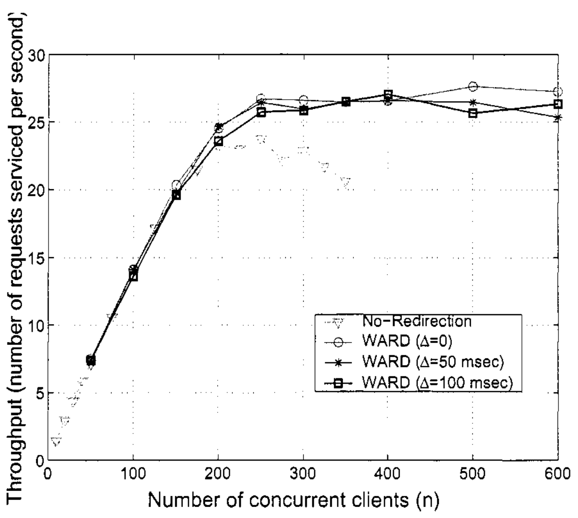


Figure 3.18 : Throughput obtained for various system configurations

duction. Figure 3.16 shows that WARD's redirection policy dispatches 24% of the database queries to the remote cluster in this case.

The performance gains of using wide area redirection increases with increasing CPU load on the local database server. For 150, 200 and 300 concurrent users, the gain achieved is 17, 40 and 54%. In this case, 150 users corresponds to a moderate load (80%) while 300 users corresponds to a heavily loaded local server (92%) with reference to the figure 3.14. Hence, to conclude, wide area redirection is of advantage for both long-term provisioning of resources when a web site wants to maintain a moderate load and for short-term bottlenecks due to flash-crowds. Similarly, the remote redirection ratio (Figure 3.16) as well as the 90-%ile response times (Figure 3.17) increase with the number of concurrent users. Therefore, WARD achieves a higher throughput than a system without redirection, as shown in Figure 3.18.

Thus, this experiment quantifies the performance gains of wide area redirection of dynamic content. Once a local server is overloaded, remote dispatching can be highly effective for network delays as high as 100 msec.

Model validation and redirection policies

Next, we validate the analytical model of Section 3.2 with testbed results for both redirection policies. In particular, we compare the redirection ratios and total response times of a system with two cluster replicas. For the model, we use Equations (3.9), (3.10) as well as Equation (3.11) from Section 3.3 with $\overline{x} = 42.9$ msec and $\sigma = 40.1$ msec, as measured on an unloaded database server in our testbed.

Figure 3.19 compares the mean query response time of the model and the implementation on a single cluster, as a function of the server load ρ. The figure indicates that the model matches the measured query response time for $\rho < 0.7$ within $\pm 10\%$. Beyond this load, the model deviates from the implementation because: (1) our M/G/1 model doesn't take read-write conflicts into account due to which queries may take longer to process that what the model predicts and (2) at high loads there are more queries and thereby greater number of conflicts.

Next, we compare the model with the two implemented redirection policies: (1) *probabilistic*, and (2) *per-query*. The per-query policy receives the CPU load measurements

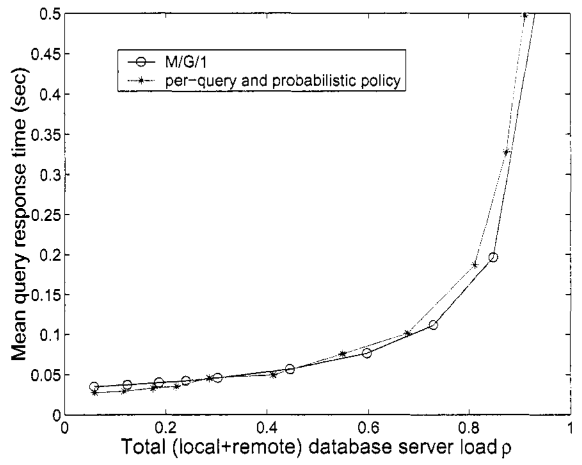


Figure 3.19 : Model verification: 1 cluster system

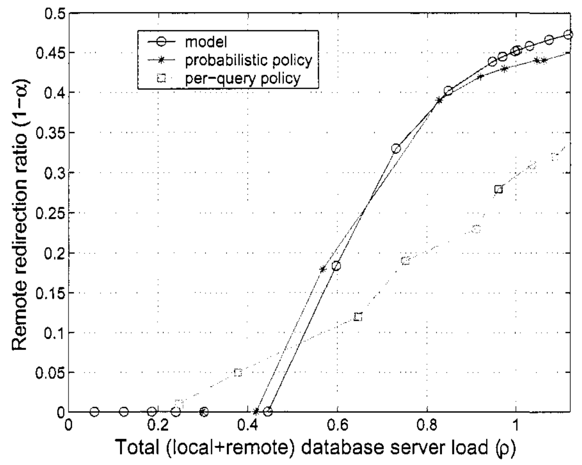


Figure 3.20 : Model verification: remote redirection ratio

every 5 sec and we set the inter-cluster latency to be 25 msec in all the experiments.

Figures 3.20 and 3.21 compare the remote redirection ratio and query response time as a function of the system load. The redirection ratios of the model and the probabilistic policy are close because this policy bases itself upon the optimal values predicted by the model. On the other hand, the per-query policy begins redirecting earlier and redirects more queries until $\rho < 0.5$ compared to both the model and

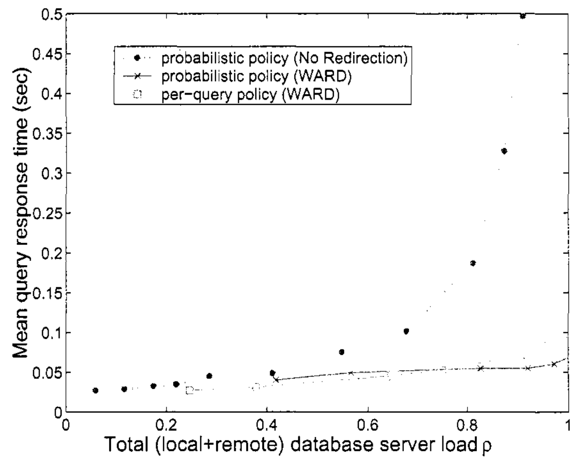


Figure 3.21 : Model verification: mean response time

probabilistic policy. The reason for this behavior is that heavy queries are more sensitive to load as shown in Figure 3.13, and hence it is of increasing value to redirect them at comparatively lower system loads. Hence a lower mean response time for the per-query policy is observed for $\rho < 0.5$ in Figure 3.21. When $\rho > 0.5$, the probabilistic policy redirects more queries than the per-query policy and hence yields lower response times. This difference can be attributed to the fact that the the measurement interval of 5 sec is too coarse grained to capture the small oscillations in CPU load. A better response time can be expected for smaller measurement intervals, but an optimal tradeoff has yet to be found between measurement accuracy and measurement overhead.

Three conclusions can be drwan from the validation of our performance model. First, the model and implementation match well for loads $\rho < 0.7$. Second, the model characterizes the basic behavior of the WARD system with a closer match for the *probabilistic* policy than the *per-query* policy largely because the model and *probabilistic* policy do not distinguish among queries. Third, measurement-based redirection policies open new optimization questions that require future work.

3.5 Related Work

Approaches to minimize web access times can be separated into different groups: resource vs. request management and, for the latter, client-side vs server-side redirection.

One approach to minimizing web access times is to ensure that enough resources are available at clusters. *Server migration* assigns servers that are unused or lightly loaded *within* a cluster to hosted applications that are suffering from high usage [31]. Server migration involves transfer of the application state from an existing server to a new server and hence migration times are on the order of 10 minutes. Therefore, server migration is a means to avoid bottlenecks over a long period of time (minutes or hours), e.g., following time-of-day patterns. In contrast, WARD is not only able to address long-term bottlenecks (at the additional redirection costs), but it is able to address short-term bottlenecks, e.g., due to flash-crowds, as well. *Server sharing*, as applied to content distribution by e.g. Villela et al. [38], is similar to server migration, except that a fraction of the resources are assigned. This option is not applicable to the architecture here because we assume that only entire servers, but not fractions of them, are assigned to individual sites. However, both server migration and sharing are orthogonal approaches to request redirection, and this dissertation advocates a combination of the mechanisms.

A significant body of research has focused on *client-side* mechanisms such as request redirection in CDNs [14, 39], server selection techniques [12, 27], caching [40], mirroring, and mirror placement [16, 41]. Such techniques are based on the premise that the network is the primary bottleneck. However, this dissertation has shown that serving dynamic content shifts the bottleneck onto the cluster. Thus, while such schemes can be applied to finding the best initial cluster, WARD's cluster-driven redirection is essential to jointly incorporate server and network latencies.

One of the most popular CDNs, Akamai [7] employs a client-side redirection policy, which integrates server-load into the server selection algorithm, using a parallel download approach. Akamai's solution for handling static content can be described as follows: First, a client is redirected to a server by a client-proxy, also known as an Akamai Data Center (DC) [42]. The Akamai DCs' form an overlay network and use ping to estimate link latencies among themselves as well as the Content Provider (CP) sites. When a client requests a name-server mapping for a particular CP site, then it is returned three routes: *direct, best two-hop and second best two-hop* to servers hosting the content. Then, the client then starts download on all three routes simultaneously and continues the download on the fastest one among them. In summary, Akamai first selects three servers on the basis of network-proximity and then accounts for the server-load as well by racing all the three routes. In contrast, in WARD, a client opens a single connection with a local cluster (with potential redirection to a remote cluster), thereby avoiding the overhead inherent in multiple connections.

Most of the current day CDN algorithms target the problem of server selection for static content, while for dynamic content they assume that most of it is cacheable and hence the algorithm for server selection for dynamic content is a direct extension of that for static. In particular, Akamai uses **EdgeSuite** [?] mechanism for serving dynamic content, by interpreting each webpage to be composed of various fragments, which could be of either static, cacheable dynamic or uncacheable dynamic types. Thus, a redirector with *EdgeSuite* enabled could select the closest server for serving the static and cacheable dynamic fragments while sending the uncacheable dynamic fragments to the origin server. The redirector then assembles the responses for all the fragments and returns it to the user. The redirectors in EdgeSuite are thus equivalent to the dispatchers in WARD, with the distinction that there is one redirector per network subnet in EdgeSuite while there is one dispatcher per origin-site cluster in WARD. Moreover, EdgeSuite assumes only one location for the origin cluster (or, server) and hence WARD can be considered as a mechanism which extends and complements EdgeSuite in selection of the best origin cluster for the uncacheable dynamic fragments.

A combination of client-side and server-side redirection is also possible and beneficial if the bottleneck is not clearly identified or varying over time. Such a combined architecture is presented by Cardellini et al. [43]. Their server-side redirection mechanism may redirect entire web requests using HTTP-redirection if the CPU utilization exceeds a certain threshold. They conclude that server-side redirection should be used selectively. In contrast, this dissertation introduces server-side redirection as a fundamental mechanism for current and future clusters. The redirection mechanism is not threshold-based, but is able to optimize cluster response times for all CPU utilization values. Moreover, Cardellini et al., design policies which consider network proximity and server load in isolation while the redirection policy proposed here integrates the two. Finally, rather than equating redirection to HTTP-redirect, WARD considers the dispatcher as a basic building block within the cluster architecture, which can redirect requests at any tier.

Chapter 4

DDoS-Shield

This chapter presents DDoS-Shield, the framework to protect a generic end-system from resource attacks. The rest of this chapter is organized as follows. Section 4.1 describes the victim, attacker, and defense models used to study layer-7 attacks. Section 4.2 describes the experimental testbed and characterize the performance impact on legitimate client sessions due to the three attack classes. Section 4.3 and Section 4.4 detail the design of the suspicion assignment mechanism and DDoS-resilient scheduler respectively as well as present their experimental evaluation. Finally, we discuss related work in Section 4.5.

4.1 Attacker, Victim and Defense System Models

This section first describes the attacker model for effecting the protocol-compliant, non-intrusive layer-7 attacks. Next, it presents the victim system over which the performance impact of these attacks is quantified. Finally, it outlines a defense model for detecting and circumventing these new attack classes.

4.1.1 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; effectively 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 that he initiates.

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. On the attacker side, we assume that the malicious client machines consume much less resources to issue requests compared to the amount of resources servers spend processing those requests. In essence, this means that the attackers are not limited by resource constraints in their ability to effect layer-7 DDoS attacks.

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

- **Request Flooding Attack:** In this class of attack, the attacker increases the rate at which each client machine issues requests.

- **Asymmetric Workload Attack:** In this class of attack, each attack session sends a higher proportion of those requests which 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 request-flooding attack as follows: (1) causes more damage by selectively sending heavier requests; and (2) is a lower-rate attack and hence involves less work on part of the attacker e.g., attacker needs fewer sessions to cause equivalent damage.

- **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 extensive monitoring and profiling. This dissertation assumes the worst case scenario that the attacker knows the full profiling data, and therefore can choose sending 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. To remove the effects of varying server loads on the server processing times, these measurements may be averaged over different times of the day.

4.1.2 Victim Model

The victim system is modeled as an e-commerce application hosted on a web cluster, which consists of multiple-tiers for processing requests, as shown in Figure 4.1. First, 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.

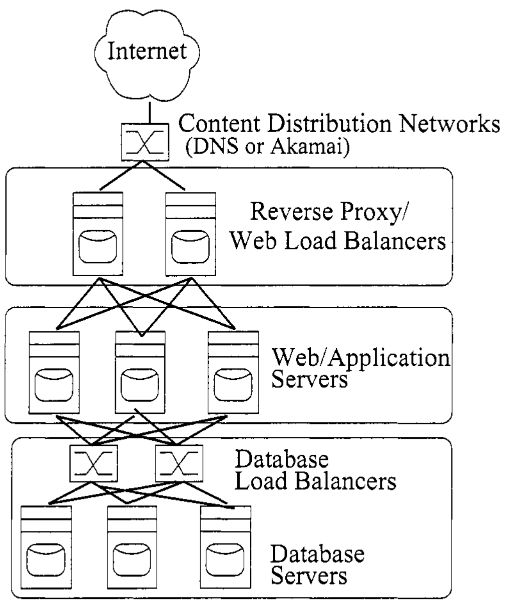


Figure 4.1 : Victim system model: web cluster hosting a web application.

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 [26]). 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. These requests are called as *dynamic requests.* Each database query emanating from a dynamic request is forwarded to a database server using a load-balancing strategy [24][37].

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 the 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 an apparent DDoS attack detected in lower layers.

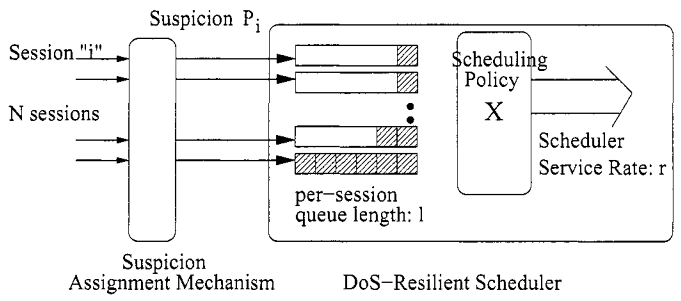

Suspicion P_i
Session "i"
N sessions
Scheduling Policy
X
Scheduler Service Rate: r
per–session queue length: l
Suspicion Assignment Mechanism
DoS–Resilient Scheduler

Figure 4.2 : Defense system model: DDoS-Shield

4.1.3 Defense Model

This dissertation introduces a counter-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 4.2 shows the architecture for DDoS-Shield consisting of: (1) Suspicion assignment mechanism which uses the session history to assign a suspicion probability p_i to every client session i as described in Section 4.3; 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 4.4.

4.2 Vulnerability to Attacks

This section characterizes the effectiveness of layer-7 DDoS attacks in overwhelming the server resources on an e-commerce application. It first quantifies 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.

4.2.1 E-Commerce Testbed

The example e-commerce application considered is an online bookstore hosted on a multi-tiered architecture consisting of three web servers and one database server. The implementation uses Apache as 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.

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 the online bookstore implementation, we profiled the processing times of individual request types to identify requests with high resource consumption on the server. Figure 4.3 shows the response times perceived across different types of requests for the online-bookstore application on our experimental system. Note that the *most expensive* request is about 8 times more expensive than the *cheapest* request. 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.

Next, we attempt to quantify the *potency* of various layer-7 DoS attacks in this

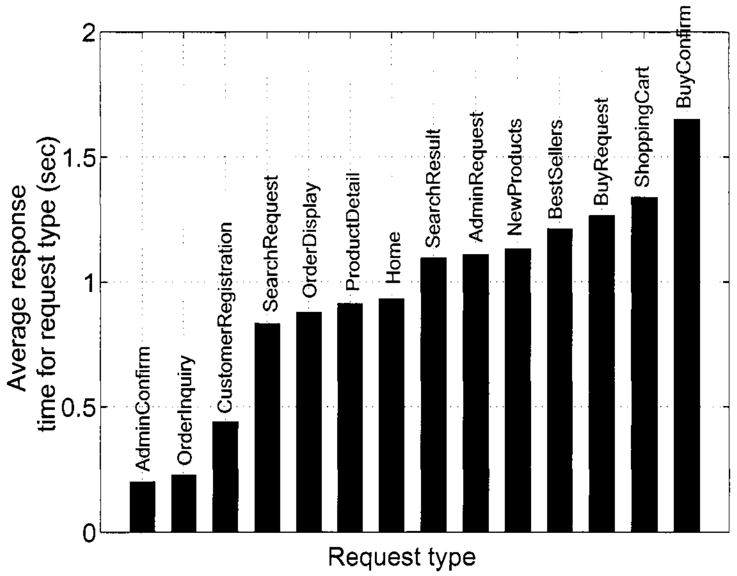


Figure 4.3 : Heterogeneity in processing times for different dynamic content requests in online bookstore application.

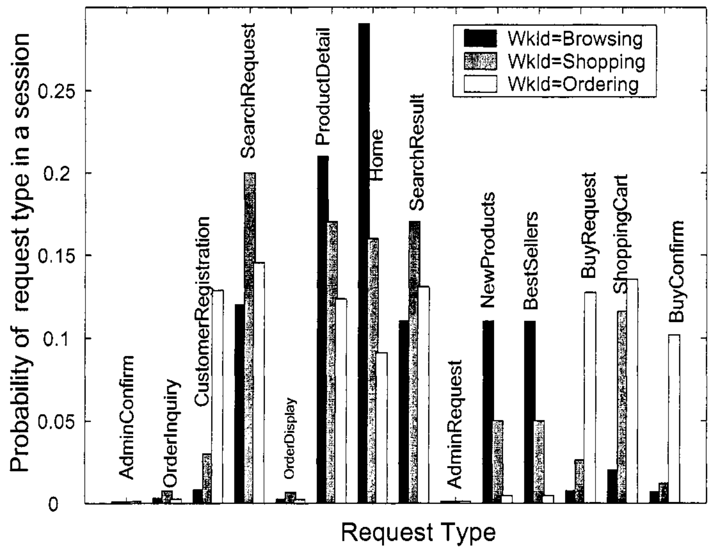


Figure 4.4 : Probability of occurrence of a request type in a client session for browsing, shopping and ordering sessions. Browsing sessions send only 5% requests for pages that involve write queries to the database server, while shopping and ordering sessions send an increasing percentage of such requests.

system. The following metrics are used 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 how much slow down a legitimate client will experience; and (3) average throughput in requests/second achieved per normal client session. The ease of mounting a layer-7 DoS attack at the attacker end point is quantified by: (1) the number of unique IP addresses; and (2) aggregate bandwidth needed to launch the attack.

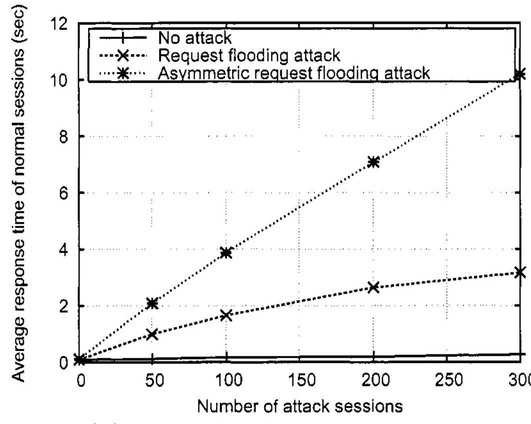
The workload of a legitimate client session is emulated using the session types shown in Figure 4.4 based on the TPC-W benchmark [22]. In particular, in each experiment, we use 100 HTTP/1.1 sessions, 33% in each of browsing, shopping and ordering profiles, to represent the legitimate client population. 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.

Each of the three types of attacks are generated 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, the think-times are decreased to 0, thereby, generating the requests as fast as possible. Second, the asymmetric workload attack is generated using one of the expensive request types. The "BestSellers" script is used for this purpose. This attack is mounted with the normal think-time of 7 seconds between requests first, and then combined 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.

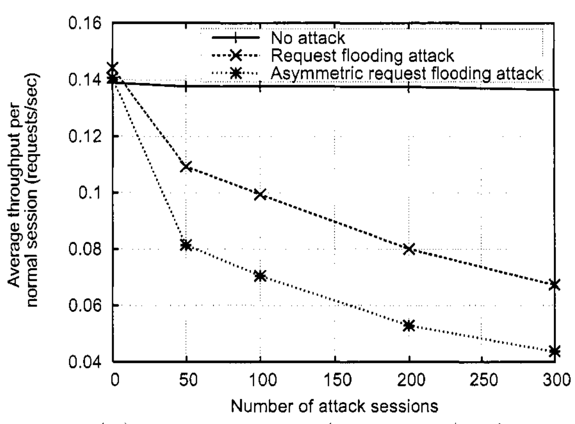
4.2.2 Attack Potency

Figure 4.5 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 [44], 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-classes as seen from Figure 4.6(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.

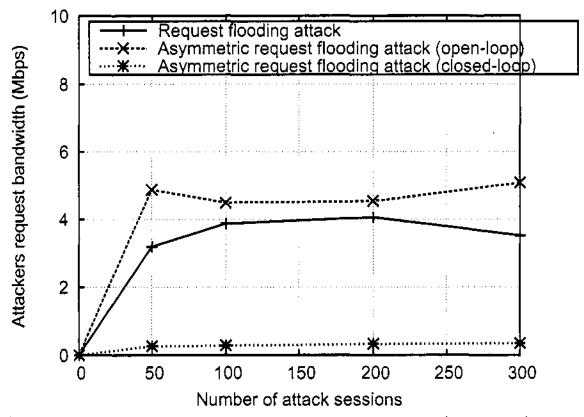
Both repeated one-shot and asymmetric attacks make the database server CPU the bottleneck, driving the CPU loads to almost 100%, in their most potent forms. However, the asymmetric attack is limited in sending a query flood towards the back-end database server since the web server serves only one request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



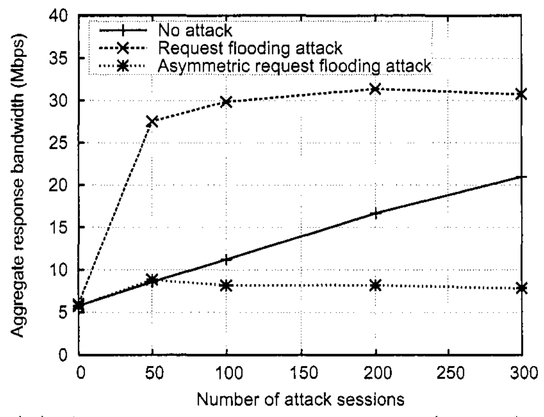
(a) Average response time



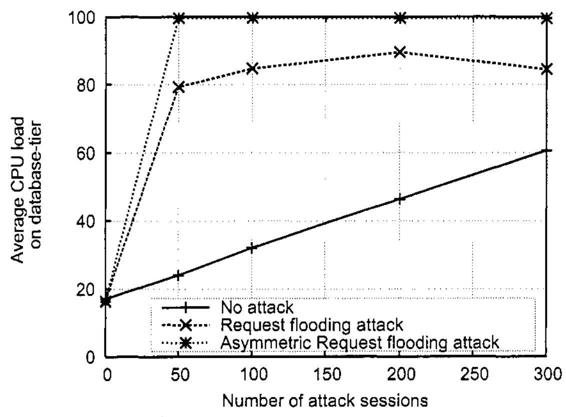
(b) Throughput (requests/sec)



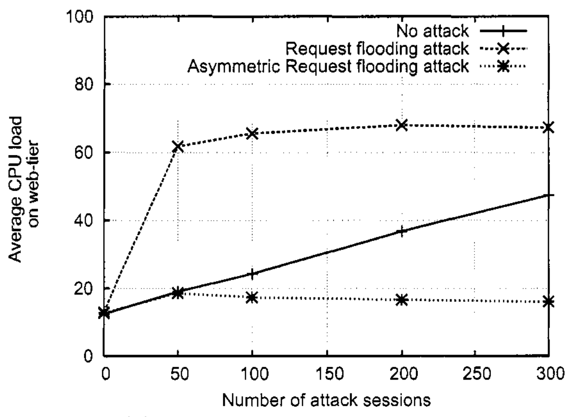
(c) Attacker request bandwidth (Mbps)



(d) Aggregate response traffic (Mbps)



(e) Database CPU load



(f) Web Server CPU load

Figure 4.5 : 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.

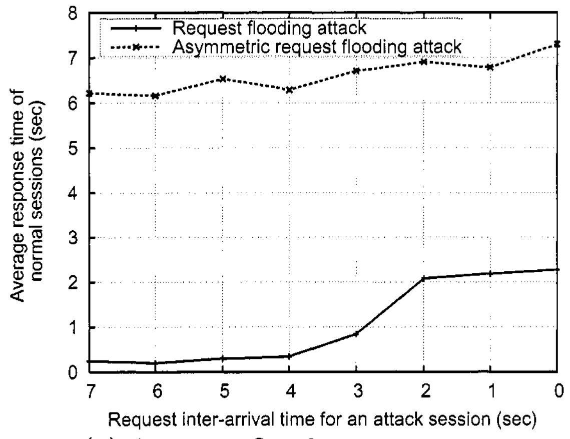
Figure 4.7 depicts the inter-arrival times between queries received at the database server. The figure shows that 90% of queries arrive within 10 msec of the previous query for the repeated one-shot attack, compared to the 80% for the asymmetric attack.

Important points to be noted are:

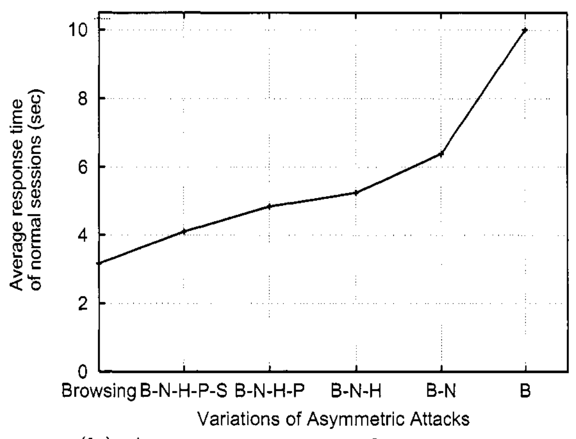
- Asymmetric request-flooding is much more potent than normal request flooding attack since it succeeds in making the database CPU the bottleneck, as observed from Figure 4.5(d). In contrast, the normal request flooding attack never makes the database CPU the bottleneck and only succeeds in increasing 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.

- Further note that during the normal request flooding attacks, the database and web server CPU loads become constant (85% and 70% respectively) instead of continually increasing upto 100%. This can be attributed as an artefact of our implementation, which consists of a limited number of TCP socket connections between the database dispatcher and database server. Thus, due to this effect along with the closed-loop behavior of Apache on sessions, the client sessions are sometimes stalled while waiting for database queries to be sent by the dispatcher to the database server. Moreover, this happens only during normal request flooding attacks and not the asymmetric flooding attacks since there is a much greater volume of requests sent in the normal flooding attack (Figure 4.5(c)) and hence the TCP sockets are more stressed.

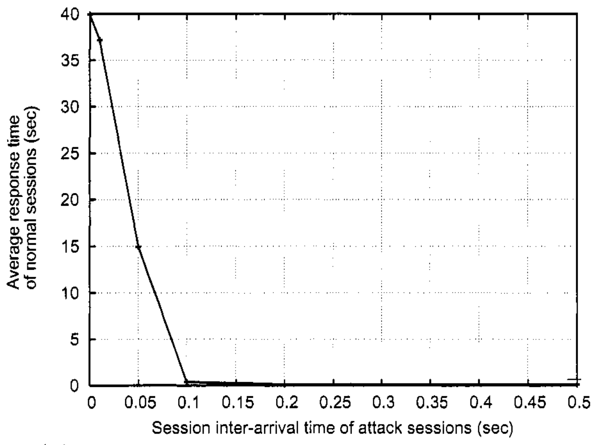
- All attacks are server attacks and the network-access link to the cluster (100 Mbps) is never overwhelmed as observed from Figure 4.5(f). The reason that



(a) Request-flooding strategies



(b) Asymmetric attack strategies



(c) Repeated one-shot attack strategies

Figure 4.6 : 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.

the aggregate download traffic saturates much before 100 Mbps is the web or database server CPUs being 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 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.
- Lastly, note that 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.

Intuitively, one may think that an attack which 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 4.5(c), than an asymmetric open loop attack sends $\approx$ 4 Mbps of request traffic compared to the much lower 0.4 Mbps by an equivalent closed loop attack, for similar damage.

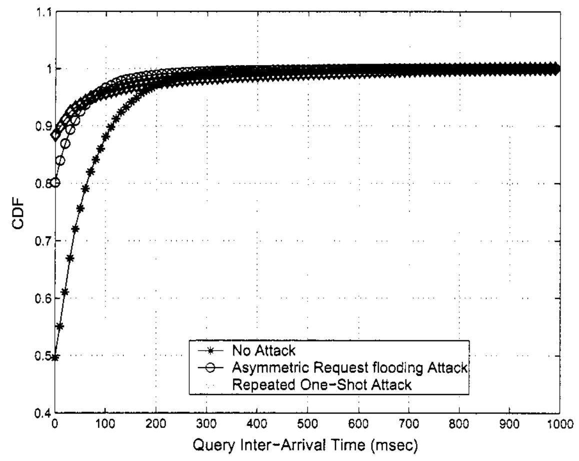


Figure 4.7 : CDF for inter-arrival times for query-arrivals at the database server for different attacks. The number of normal browsing sessions is 100 in each scenario. The no attack case has 0 attackers, while the asymmetric request-flooding corresponds to 300 attack sessions with request think-time of 0 seconds. The repeated one-shot attack corresponds to an attack session being opened immediately (0) seconds after closing the previous session.

4.2.3 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 4.6(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 4.3, suppose an attacker picks up the following request types in decreasing order of processing times: **B**estSellers > **N**ewProducts > **H**ome > **P**roductDetail > **S**earch. The following attacker strategies are investigated: (1) *B*: 100% BestSellers requests, (2) *B-N*: equal number of BestSellers and NewProducts requests and similarly, (3) *B-N-H*, (4) *B-N-H-P*, (5) *B-N-H-P-S*. Figure 4.6(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 4.6(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.

4.3 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.

The suspicion-assignment problem is formulated by first performing measurements to characterize the set of distributions that define legitimate behavior. The suspicion of a session is then calculated on the basis of the probability that it was generated from one of the legitimate distributions. Recall from Section 4.2 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, this section describes the offline phase in which the legitimate client behavior profiles are built from system logs. Next, it describes suspicion assignment techniques corresponding to each of the three kinds of deviations from normal behavior, followed by an algorithm to combine their outputs. Finally, this section is concluded by presenting testbed results to evaluate the performance of our suspicion assignment techniques.

4.3.1 Legitimate Client Profiles

In this phase, information is extracted from the system logs to build profiles for legitimate client behavior with respect to each of the following workload parameters: 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.

This dissertation assumes stationarity in the system logs and hence the behavior profiles extracted are assumed to be invariant with respect to either time or day effects. However, e-commerce traffic can be expected to vary across either the time-of-day, with more traffic expected during the day than night or the day-of-year, with more traffic expected during Christmas or other shopping seasons. Thus, while designing more sophisticated systems, a set of legitimate profiles can be built corresponding to time-of-day and day-of-year.

Further, this dissertation assumes that the system logs used for building the profiles are not influenced by attackers. However, note that a wily attacker may inject a certain "attacker favorable" traffic pattern into the system logs with the objective of swaying the system's suspicion assignment mechanism in to his favour. This may be done by sending requests following a certain pattern at a rate that is below the radar of the suspicion assignment mechanism, over an extended period of time. These attacks can be countered either by using a statistically more-significant number of observations to build the profiles or by using stricter confidence intervals. However, by assuming unadulterated system logs, these attacks are rendered out of scope from this dissertation.

- **Session Inter-Arrival Distribution:** We extract the session inter-arrival times between consecutive sessions to obtain the distribution A by which these

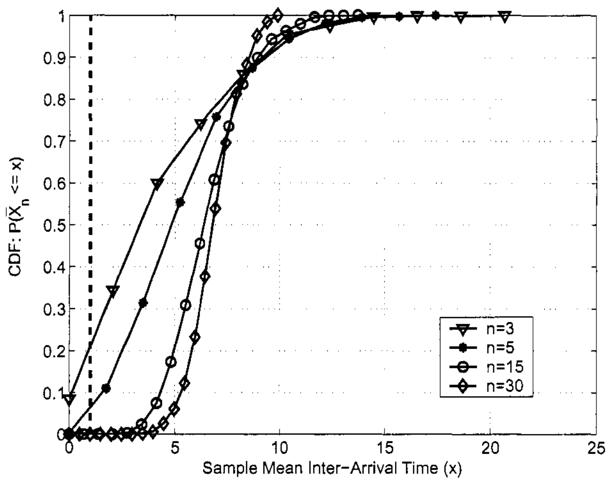


Figure 4.8 : Cumulative Distribution Function for request inter-arrival time: $P(\overline{X_n} \leq x)$ with varying session lengths or sample sizes 'n'.

times are distributed. In particular, due to our workload generator, this distribution A is exponential with mean 0.2 seconds.

- **Request Inter-Arrival Distribution:** First, we extract all sessions of length smaller than or equal to n requests. Next, we obtain the request inter-arrival times between consecutive requests across all these sessions, to obtain a sample distribution of request inter-arrival time: X_n. This is done for all values of $n = [2 - 60]$ to obtain several sample distributions, as shown in Figure 4.8. Observe that with increasing sample sizes n, the distribution 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), a standard centroid-clustering algorithm [45][46] can be used 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 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 their 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 [45] 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 *BestSellers.* Sessions are clustered into 3 session types, identified as: *browsing*, *shopping* and *ordering* (see Figure 4.4).

4.3.2 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 session inter-arrival times larger 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, the latter half of this section, presents an algorithm to reduce the performance impact due to these false-positives.

4.3.3 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 4.8 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, asymptotically getting closer to 0.99 with further observations.

4.3.4 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 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, a framework for *soundness* of a workload suspicion measure is developed, using which a measure can ensure consistency in assignment of suspicions across workload deviations.

A desirable distance metric disassociates length 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. This dissertation considers two candidate distance metrics to illustrate the properties: *Kullback Leibler (KL)* distance metric from information-theory [47] and a metric developed here, called *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 $\sum_{i=1}^{r} G(a_i) = 1$.

Distance Metrics

Definition 1 *The KL distance between the session type $T(s)$ characterizing a session s and the ideal distribution G that it is expected to be derived from is defined as:*

$$KL(T(s)||G) = \sum_{a_i \in \sigma} N(a_i) \log \frac{N(a_i)}{G(a_i)}. \tag{4.1}$$

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:*

$$RF(s||G) = \frac{res}{gcf} \tag{4.2}$$

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). The case where there is no good subtype, i.e., $gcf = 0$ can be handled separately.

Observe that both KL-distance and RF-distance have the properties desirable in a distance metric. Let's illustrate with an example on two request classes: $\sigma = 0, 1$ and ideal distribution $G : Bernoulli(\sigma), (p_0, p_1) = (0.5, 0.5)$. If the session s has the same type as the distribution G, then both distance metrics assign distance 0. Sessions of type $(0.8, 0.2)$ 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 than that assigned to sessions of type $(0.9, 0.1)$, in which case the KL-distance is 0.368 and RF-distance is 4.

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.

  $$T(s) = G \implies f(s) = 0 \;\; \forall n \in [1, \infty) \tag{4.3}$$

  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:

  $$n_1 = n_2, \;\; d_1 > d_2 \implies f(s_1) > f(s_2) \tag{4.4}$$

  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:

  $$T(s_1) = T(s_2) \neq G, \;\; n_1 > n_2 \implies f(s_1) > f(s_2) \tag{4.5}$$

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

There are several possible metrics f which satisfy the properties of soundness. To establish the veracity of the properties, let's first consider an intuitive *enumeration* metric to assign suspicion numbers.

Enumeration Metric

This method exhaustively enumerates all possible session types of length n and sorts these types by their probability of occurrence. Using basic combinatorial theory, the probability of occurrence of a session type $T(s)$ assuming generating distribution G is given as the product of the number of sessions/sequences which have this type (given by multinomial coefficient) and the probability of observing any one of these sessions (given by product of probability of observing alphabet a_i $n(a_i)$ times).

$$\begin{aligned} P^n(T(s)) &= \frac{n!}{n(a_1)!n(a_2)!...n(a_r)!}P(T(s)|G). \\ P(T(s)|G) &= G(a_1)^{n(a_1)}...G(a_r)^{n(a_r)}. \end{aligned} \tag{4.6}$$

Definition 3 *Hence, by the enumeration metric, the suspicion associated with session s of type $T(s)$ is defined as the probability of finding other types of same length n which have a higher probability of occurrence.*

$$f_{enum}(s) = \sum_{\tau_i^n} P^n(\tau_i^n),\ where\ \tau_i^n : P^n(\tau_i^n) > P^n(T(s)) \tag{4.7}$$

The enumeration metric satisfies all the properties of soundness.

Proof:

- *Zero-distance Property:* Note that if a session s has the same distance as the ideal distribution G, then amongst all the sessions of same length n, it will be assigned the highest probability of occurrence: $P^n(T(s))$. Thus, from Definition 3, it will be assigned a suspicion of 0, since there are no types with a higher probability of occurrence than $P^n(T(s))$.

- *Distance-Proportionality Property:* Method of Types [48] provides an asymptotic bound on the occurrence probability $P^n(T(s))$ as follows:

$$P^n(T(s)) \approx e^{-nKL(T(s)||G)}. \tag{4.8}$$

  Using this bound, if two sessions have same length n, then the session with larger KL-distance would have lower occurrence probability, and hence by Definition 3, a higher suspicion.

- *Length-Proportionality Property:* Using the same bound as in Equation 4.8, it is easy to verify that enumeration also satisfies the length-proportionality property. Thus, if two sessions have the same KL-distance from legitimate distribution, then the longer session would have lower occurrence probability and hence by Definition 3, a higher suspicion.

■

Note that these suspicion values are asymptotically correct, i.e.,

$$\begin{aligned} \lim_{n\to\infty} f(T(s)) &= 0 \;\; if \;\; T(s) = G \\ &= 1 \;\; if \;\; T(s) \neq G \end{aligned} \tag{4.9}$$

Moreover, by associativity of the properties of distance- and length-proportionality, the most-anomalous type, i.e., the one with the largest KL-distance converges to 1 the earliest, followed by the 2nd-most anomalous session-type and so on.

The method of enumeration is accurate but computationally infeasible. Even for the online bookstore benchmark, where the number of request-classes is $r = 14$, to consider sessions of length $n = 30$, the number of types needed is a very large polynomial $= \frac{43!}{30!13!} = 3.65 * 10^{10}$.

There are several possible measures f which satisfy the properties of soundness, even though the suspicion measures assigned by them may not be accurate as the enumeration measures. 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.

Length Distance Product (LDP) Measure

Definition 4 *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 dissertation, we get the following equivalent measure definitions:*

$$\begin{aligned} f_L^{KL}(s) &= n\,KL((T(s)||G) \\ f_L^{RF}(s) &= n\,RF(S||G) \end{aligned} \tag{4.10}$$

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:

$$f_L^{KL}(s) = n \min_j (KL(\,(T(s)||G_j)\,))$$
$$f_L^{RF}(s) = n \min_j (RF(\,(s||G_j)\,)) \tag{4.11}$$

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 as follows:

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

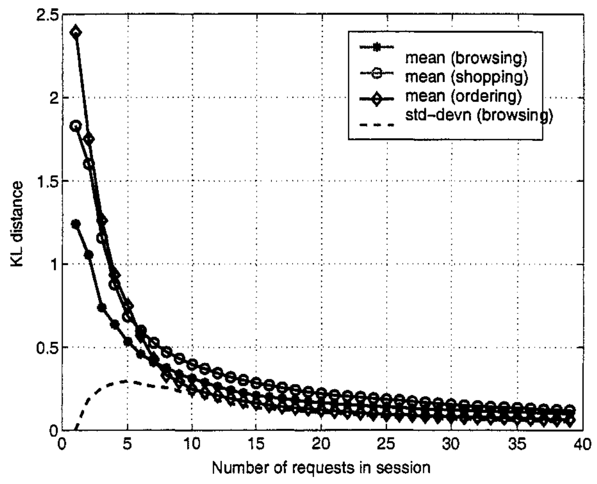
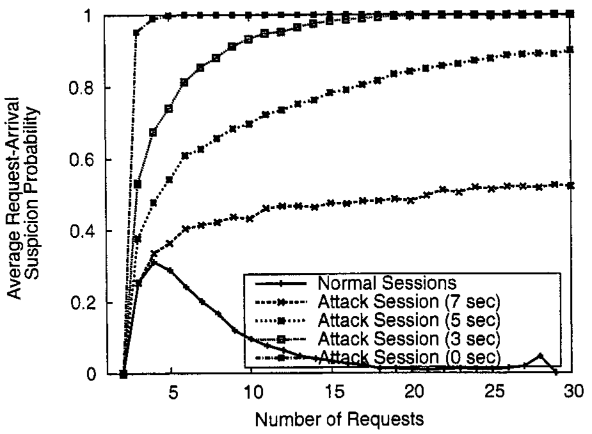
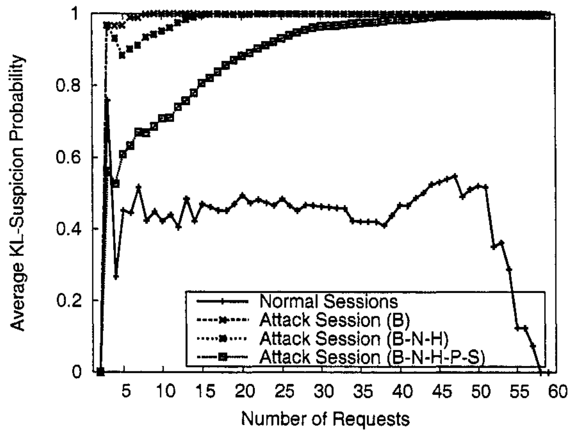


Figure 4.9 : Mean of KL-distance of "browsing", "shopping" and "ordering" sessions with increasing sample-sizes n.

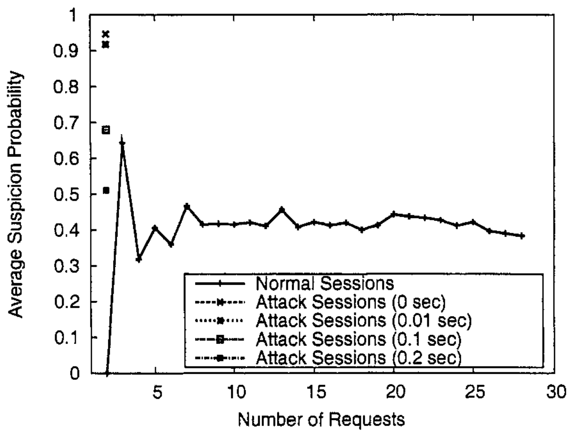
Our online bookstore implementation consists of three ideal distributions: $G_{browsing}$, $G_{shopping}$ and $G_{ordering}$ as shown in Figure 4.4. Figure 4.9 shows the average KL-distance of a browsing session with respect to $G_{browsing}$ with increasing number of requests n. Note that the KL-distance of a legitimate session with its ideal distribution converges to 0 with increasing number of requests n.



(a) Request flooding attacks



(b) Asymmetric attacks



(c) Repeated one-shot attacks

Figure 4.10 : Average suspicion probability of a normal or attack session with increasing number of requests seen in the session.

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: $f^n_{request}(s)$;and $f^n_L(s)$. Thus, using a suspicion weighting parameter $0 \leq \beta \leq 1$, we define the net suspicion measure $f^n(s)$ as follows:

$$f^n(s) = f_{session}(s) * (\beta\, f^n_L(s) + (1-\beta)\, f^n_{request}(s)) \tag{4.12}$$

Note that net suspicion is contained 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 were 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 and even if it goes away after sending one request, it would have been serviced with lower quality.

- 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. Let's 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.

Performance of Suspicion Assignment

We next provide numerical results for the performance of the suspicion assignment techniques on attacks launched against our online-bookstore implementation. Figure 4.10 shows the behavior of suspicion measure with 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. The following observations can be made:

- In either request flooding or asymmetric attacks, the attack sessions can be distinguished from normal sessions after 4 requests on an average. Thus, a counter-DDoS policy could start punishing attack sessions from very early on.
- Normal sessions converge to suspicion of 0 with respect to the request-arrival and workload after 17 and 57 requests respectively as seen from Figure 4.10(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 are needed to detect it with certainty.
- An asymmetric attack session sending BestSellers requests is detected with suspicion probability of 1.0 after 8 requests on an average. Moreover, if the attacker morphs his 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.

4.4 Scheduler Design for DDoS-Shield

This section presents the DDoS-resilient scheduling policy of *DDoS-Shield*, which combines the continuous measure of suspicion assigned by the suspicion mechanism with the current system workload to decide whether and when a session is allowed to forward requests (see Figure 4.2). The DDoS-resilient scheduler is integrated into the reverse proxy, and can thus intercept requests belonging to malicious sessions much before they overwhelm the system resources.

The maximum aggregate rate at which the scheduler forwards requests to the web cluster is a configurable parameter called the *scheduler service rate*: r requests/second. 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. Whenever, the current output rate is less than the scheduler service rate, using a scheduling policy, 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 having 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 4.11 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[1]. 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.

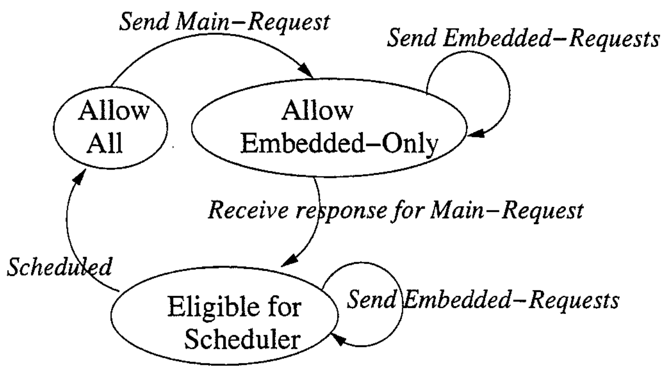


Figure 4.11 : State Diagram for a session in scheduler queue

[1]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.

4.4.1 Scheduling Algorithms

The following scheduling algorithms are considered:

- **First-Come First-Serve (FCFS) Scheduler:** The FCFS scheduler schedules the session with the earliest arrived HoL request from amongst all the eligible sessions.

- **Round-Robin Scheduler:** After scheduling an eligible session once, the Round-Robin scheduler schedules it again only after it becomes eligible again and after all other eligible sessions have been scheduled before it.

- **Shortest Job First (SJF) Scheduler:** The scheduler selects the session whose HoL request has the lowest estimated service time. The service time of a request is estimated on the basis of its request type as shown in Figure 4.3.

- **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:

  $$\min \sum_{i=1}^{N} (1 - p_i)(d_i) \tag{4.13}$$

  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 $1 - p_i$:
$$\frac{r_i}{r_j} = \frac{1 - p_i}{1 - p_j} \tag{4.14}$$

Note that the FCFS, Round-Robin and SJF schedulers are agnostic to suspicion probabilities, and hence they serve as baseline for performance evaluation against the scheduling policies: *LSF* and *PSS* which use suspicion probabilities.

4.4.2 Online Rate-Setting Algorithm

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: $r_i = (1 - p_i) r_{0.95}$.

4.4.3 Performance Evaluation

First, we establish through experiments that to be effective, a counter-DDoS mechanism needs both scheduling and rate limiting. We compare the suspicion-aware scheduling policies, LSF and PSS against the suspicion-agnostic policies, Round Robin and FCFS and also vary the scheduler service rates in each experiment. 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.

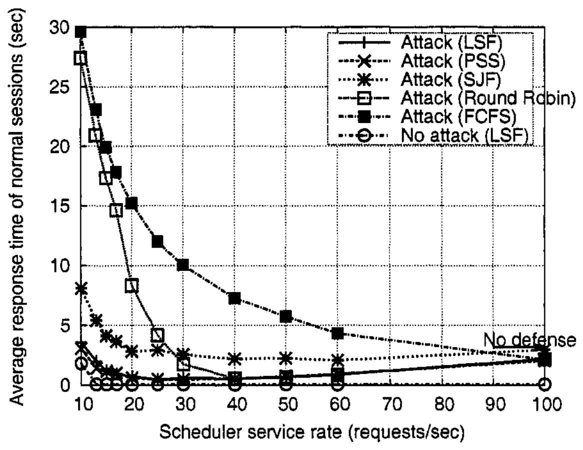


Figure 4.12 : Effect of various scheduling policies and scheduler service rates on 100 normal browsing sessions in the presence of 300 request flooding attack sessions.

Request-flooding Attack

Let's first consider the most potent request-flooding attack using 300 attack sessions on 100 normal sessions, shown in Figure 4.12. The main conclusions we draw are:

- The combined strategy of using scheduling along with a rate limiter is quintessential to DDoS-Shield. Thus, the best performance is obtained on using a LSF or PSS scheduler with scheduler service rate set in the range [15 – 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), and DDoS-Shield's performance of 0.5 seconds compares favourably with the lower-bound performance under no attack of 0.1 seconds.

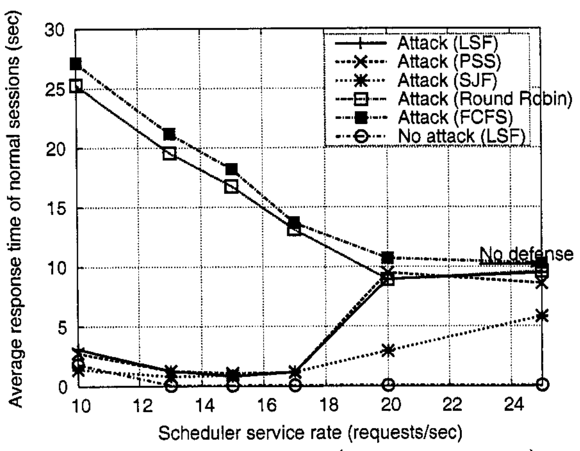
- The LSF and PSS schedulers perform the best, with LSF slightly better. The Round-Robin and FCFS schedulers are agnostic to suspicion probabilities and still admit many malicious sessions leading to significantly lower performance.

- SJF is still better than FCFS and Round-Robin, however, worse than the suspicion aware schedulers. The reason being that in request flooding attack, requests of type (hence size) same as the legitimate client sessions are used and hence admitting requests on the basis of their estimated size does not necessarily protect from the attack.

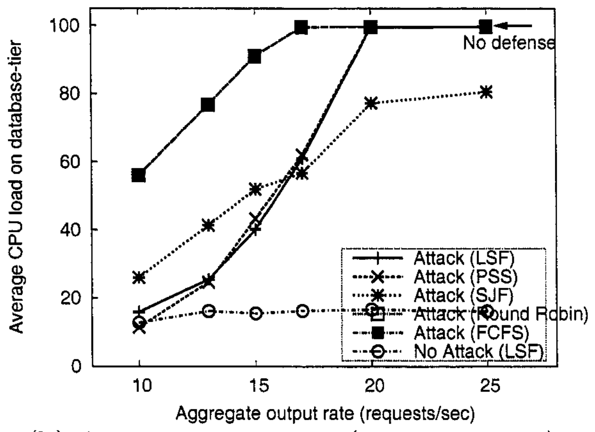
- However, even the performance achieved by using the best schedulers (LSF, PSS) degrades, improves and again degrades with increasing service rates. This is on account of the fact that at lower scheduler service rates, normal sessions are not allocated enough service, while at higher rates, many attack sessions are admitted.

- All scheduling algorithms converge to an average response time of 2.2 seconds at service rates greater than 100 requests/second. Even then, their performance compares better than no defense (FCFS, $l = \infty$) performance of 3 seconds, since due to smaller session queues ($l = 100$), they drop several requests belonging to attack sessions.

- When online rate-setting algorithm is used along with the LSF scheduling policy, we get similar performance improvements at around 0.5 seconds. The average service rate set by online rate setting was 17 requests/second when we used $\alpha = 0.3$ and updated the rate every 10 seconds.



(a) Asymmetric attack (response time)



(b) Asymmetric attack (DB CPU load)

Figure 4.13 : Effect of various scheduling policies and scheduler service rates on 100 normal sessions under most-potent 300 asymmetric attack sessions.

Asymmetric Attack

DDoS-Shield improves the performance under the most-potent asymmetric attack from 10 seconds to 0.8 seconds, as seen from Figure 4.13(a). 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 (FCFS and Round-Robin). The reason is that at high service rates, a slight increase in admittance of attack sessions drives the database server CPU loads to as high as 100%, as depicted in

Figure 4.13(b).

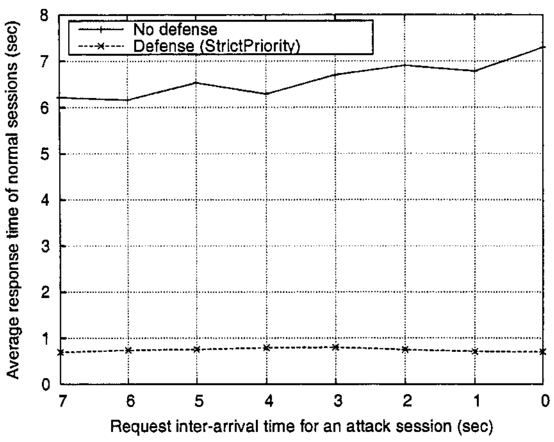
Another important observation derived from Figure 4.13(a) is that our suspicion aware schedulers perform as good as the baseline SJF scheduler at service rates lower than 17 requests/sec. This is expected, since the asymmetric flooding attack works by sending heavy requests and hence scheduling in the light requests first is an effective strategy to protect from these attacks. However, at higher service rates, SJF performs better than the suspicion aware schedulers, which is a side-effect of more false-negatives being admitted by the suspicion aware schedulers. Although SJF performs the best under certain service rates against the most-potent asymmetric attacks, we do not expect the same behavior under the less-potent or morphed asymmetric attacks. For instance, under an asymmetric attack which sends the lightest requests, say SearchRequest (Figure 4.3), the SJF policy would admit them before the legitimate client requests, and hence can be expected to perform worse in comparison with our suspicion aware schedulers.

Repeated One-shot Attack

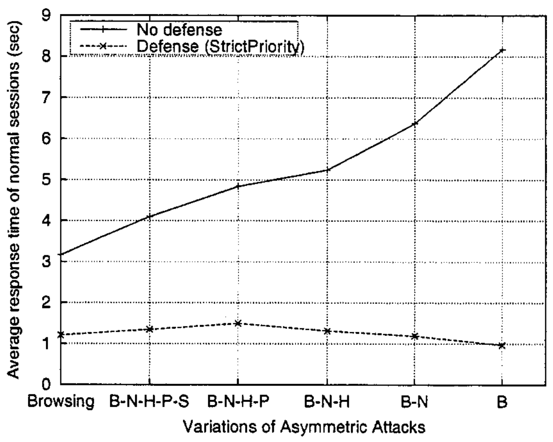
Similarly, for repeated one-shot attacks, DDoS-Shield improves the performance under the most-potent attack (inter-session time=0 seconds) from 40 seconds to 1.5 seconds. The best performance is achieved using LSF scheduler at scheduler service rates around 15 requests/second.

Low Potency Attacker Strategies

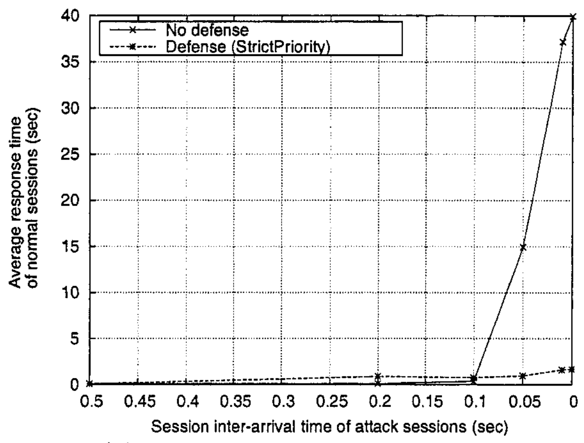
Recall from discussions in Section 4.3 that lower potency attacks are more difficult to detect than high potency attacks. Hence, to demonstrate the efficacy of DDoS-Shield in thwarting low potency attacks, we evaluate the performance under varying request flooding and varying asymmetric attack strategies. Using the scheduling



(a) Variations in think-time



(b) Variations in workload profile



(c) Repeated one-shot attacks

Figure 4.14 : Performance of DDoS-Shield on protecting 100 normal sessions when under attack by lower-potency attacker strategies. DDoS Shield uses LSF scheduler and scheduler service rate set at 15 requests/second.

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 lower potency attacks as well as high potency ones, is on account of the suspicion assignment mechanism being able to differentiate between legitimate and malicious sessions from very early on. Recall from Figure 4.10(a) that even though lower-rate or lower-intensity attacks are detected with certitude much later than their high potency counterparts, on an 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.

Summary of Results

The main conclusion drawn from our experimental investigation of DDoS-Shield is that to be effective a counter-mechanism must employ both rate-limiting and scheduling. Furthermore, suspicion aware schedulers are more effective than suspicion agnostic schedulers in admitting the legitimate requests and delaying the malicious ones. Lastly, DDoS-Shield with the suspicion aware policies protects under both the most-potent as well as the less-potent and morphed attack scenarios, due to the suspicion assignment mechanism being able to quickly differentiate between legitimate and malicious sessions.

4.5 Related Work

CERT [49] 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 dissertation 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 [50][21].

4.5.1 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 [51–53]. For example, the anomaly detection system in [54] 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 dissertation 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 [55]. While the attacks in this dissertation 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 is identified in [30]: (1) DoS event is due to increase in the request rates for a small group of clients while flash crowds are due to increase in number of clients; and (2) DoS clients originate from new client clusters [2] 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 dissertation since (1) it is difficult to associate 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 [56][57] 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.

4.5.2 Counter-DDoS Mechanisms

Kandula et. al. in [50] design a system to protect a web cluster from DDoS attacks by (1) designing a probabilistic authentication mechanism using CAPTCHAs

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

(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. 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 automated methods [58]. 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. However, Turing tests are complementary to DDoS-Shield and their result could be used as an input to decide the seed suspicion of a session, where sessions which pass the test start with a lower suspicion and vice-versa.

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 [54] or rate-limited [59]. Class-based queuing scheme used in [60] uses a load balancer to block or limit the service to client IP addresses depending on their bandwidth consumption patterns. Similarly, probabilistic queuing scheme in [61] 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 [62]. These techniques are all geared towards preventing large 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.

Chapter 5

Conclusions

This dissertation first characterized the reasons for disruption and performance degradation in web services as due to traffic overload conditions, and then designed a web hosting architecture using which web services can optimize client access times despite the presence of these conditions. The major contributions and findings of this dissertation are summarized below.

Standard overload conditions such as flash crowd arrivals and time-of-day variations increase client access latencies much larger than the average user patience while downloading content estimated at around 5 seconds. As its first contribution, in Chapter 3, this dissertation presented WARD, a framework for wide-area redirection of dynamic content requests. The objective of WARD is to minimize the end-to-end latency of dynamic content requests by jointly considering network and server delays. Thus, WARD can effectively multiplex resources across the entire cluster grid, thereby avoiding performance degradation in the presence of the standard overload conditions. Chapter 3 further described an analytical model and a proof-of-concept implementation which demonstrated significant reductions in average request response times. For example, for a cluster implementation running an e-commerce site and serving 300 concurrent users, WARD was shown to reduce average response times by 54% from 5 sec to 2.3 sec, while redirecting requests away to a cluster 50 msec away. Moreover, the analytical model predicted significant performance improvements if the complexity of dynamic content processing in web requests would increase in terms of either the mean or the variance in service times. WARD is especially suited to prevent increased response times due to short-term bottlenecks, e.g., caused by flash crowds. If the latency costs of redirection are not excessively high, WARD can also be used to exploit long-time-scale trends such as time-of-day driven workloads, and thereby avoid expensive over-provisioning of clusters. Finally, WARD is an orthogonal solution to client-side redirection and server migration policies and can therefore be seamlessly integrated with such approaches.

DDoS attackers have been amassing vast amount of resources into server farms also popularly known as botnets. Moreover, attacks are becoming more sophisticated and consequently more difficult to detect. Thus, as its second contribution, this dissertation in Chapter 4, explored the vulnerability of systems to one such class of sophisticated attacks. These attacks are protocol-compliant as well as non-intrusive and mimic legitimate clients of the service with the intention of overwhelming the system's resources. Thus, this dissertation presented a framework to classify 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 exploited. Since, these resource attacks are un-detectable via sub-layer-7 techniques, this dissertation developed DDoS-Shield, a counter-mechanism which assigns suspicion probability to a session in proportion to its deviation from legitimate behavior and uses a DDoS-resilient scheduler to decide whether the session is serviced and when. An example web application hosted on an experimental testbed, was used to demonstrate the potency of these attacks as well as the efficacy of DDoS-Shield in preventing them. While, the attacks on our online bookstore implementation stressed the server resources the most, our attacker model as well as counter-mechanism are general and easily extensible to other resources.

Bibliography

[1] P. Coelho, *The Alchemist.*

[2] A. Odlyzko, “Data networks are mostly empty and for good reason,” *IT Professional*, vol. 1, no. 2, pp. 67–69, Mar. 1999.

[3] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot, “Packet-level Traffic Measurement from the Sprint IP Backbone,” *IEEE Network Magazine*, 2003, To be published.

[4] M. Arlitt, D. Krishnamurthy, and J. Rolia, “Characterizing the scalability of a large web-based shopping system,” *ACM Transactions on Internet Technology*, vol. 1, no. 1, pp. 44–69, Aug. 2001.

[5] M. Arlitt and C. Williamson, “Internet web servers: Workload characterization and performance implications,” *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, Oct. 1997.

[6] “Slashdot,” http://www.slashdot.com.

[7] “Akamai,” http://www.akamai.com.

[8] “Digital Island,” http://www.digitalisland.net.

[9] “Mirror Image Internet,” http://www.mirror-image.com.

[10] J.D. Guyton and M.F. Schwartz, “Locating nearby copies of replicated internet servers,” in *Proceedings of ACM SIGCOMM’95*, Cambridge, MA, Aug. 1995.

[11] A. Shaikh, R. Tewari, and M. Agrawal, "On the effectiveness of dns-based server selection," in *Proceedings of IEEE INFOCOM'01*, Anchorage, AK, Apr. 2001.

[12] Z. Fei, S. Bhattacharjee, E.W. Zegura, and M.H. Ammar, "A novel server selection technique for improving the response time of a replicated service," in *Proceedings of IEEE INFOCOM'98*, San Francisco, CA, Mar. 1998.

[13] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *ACM Symposium on Theory of Computing*, May 1997, pp. 654–663.

[14] L. Wang, V. Pai, and L. Peterson, "The effectiveness of request redirection on cdn robustness," in *Proceedings of OSDI'02*, Boston, MA, Dec. 2002.

[15] M. Karlsson and M. Mahalingam, "Do we need replica placement algorithms in content delivery networks," in *7th International Workshop on Web Content Caching and Distribution (WCW)*, August 2002.

[16] Y. Chen, R.H. Katz, and J.D. Kubiatowicz, "Dynamic replica placement for scalable content delivery," in *Proceedings of IPTPS'02*, Cambridge, MA, Mar. 2002.

[17] L. Qiu, V. Padmanabhan, and G. Voelker, "On the placement of web server replicas," in *Proceedings of IEEE INFOCOM'01*, Anchorage, AK, Apr. 2001.

[18] S. Mittal, "A consistent and transparent solution for caching dynamic web content," in *Masters Thesis*, Rice University, Houston, TX, May 2005.

[19] President's Information Technology Advisory Committee, "Cyber security: A crisis of prioritization," www.nitrd.gov/pitac/reports/index.html.

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

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

[22] "TPC-W: Transaction Processing Council ," http://www.tpc.org.

[23] "NISTNET: Network Emulation," http://snad.ncsl.nist.gov/itg/nistnet/.

[24] S. Ranjan, R. Karrer, and E. Knightly, "Wide area redirection of dynamic content in internet data centers," in *Proceedings of Infocom*, HongKong, 2004.

[25] 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," Tech. Rep. TR-02-391, Rice University, Feb. 2002.

[26] 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.

[27] R.L. Carter and M. Crovella, "Server selection using dynamic path characterization in wide-area networks," in *Proceedings of IEEE INFOCOM'97*, Kobe, Japan, Apr. 1997.

[28] "Cisco Distributed Director," http://www.cisco.com/warp/public/cc/pd/cxsr/dd/index.shtml.

[29] V. N. Padmanabhan and K. Sripanidkulchai, "The case for cooperative networking," in *1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, Cambridge, MA, Mar. 2002.

[30] 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*. May 2002, pp. 252–262, IEEE.

[31] S. Ranjan, J. Rolia, H. Fu, and E. Knightly, "Qos-driven server migration for internet data centers," in *Proceedings of IWQoS'02*, Miami, FL, May 2002.

[32] "The Apache Software Foundation," http://www.apache.org.

[33] "PHP Scripting Language," http://www.php.net.

[34] "MySQL Database Server," http://www.mysql.com.

[35] C. Amza, A. Cox, and W. Zwaenepoel, "Scaling e-commerce sites," Tech. Rep. TR-02-390, Rice University, Houston, TX, Feb. 2002.

[36] C. Amza, E. Cecchet, A. Chanda, A. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel, "Specification and implementation of dynamic content benchmarks," in *Proceedings of the 5th IEEE Workshop on Workload Characterization (WWC-5)*, Austin, TX, Nov. 2002.

[37] 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, Mar. 2003.

[38] D. Villela and D. Rubenstein, "Performance analysis of server sharing collectives for content distribution," in *Proceedings of IWQoS'03*, Monterrey, CA, June 2003.

[39] J. Kangasharju, K.W. Ross, and J.W. Roberts, "Performance evaluation of redirection schemes in content distribution networks," *Computer Communications*, vol. 24, no. 2, pp. 207–214, Feb. 2001.

[40] D. Karger, A. Sherman, A. Berkhemier, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi, "Web caching with consistent hashing," in *Eighth International World Wide Web Conference*, May 1999.

[41] S. Jamin, C. Jin, A.R. Kurc, D. Raz, and Y. Shavitt, "Constrained mirror placement on the internet," in *Proceedings of IEEE INFOCOM'01*, Anchorage, AK, Apr. 2001, pp. 31–40.

[42] "Why traditional routing is not always the best method," http://www-math.mit.edu/ steng/18.996/lecture9.ps.

[43] V. Cardellini, M. Colajanni, and P. S. Yu, "Geographic load balancing for scalable distributed web systems," in *Proceedings of MASCOTS'00*, San Francisco, CA, Aug. 2000.

[44] 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.

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

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

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

[48] I. Csiszar, "The method of types," *IEEE Transactions on Information Theory*, vol. 44, pp. 2505–2523, 1998.

[49] "http://www.cert.org," 2005.

[50] 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.

[51] L. Ricciulli, P. Lincoln, and P. Kakkar, "TCP SYN flooding defense," in *CNDS*, 1999.

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

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

[54] 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.

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

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

[57] 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.

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

[59] 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.

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

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

[62] 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.