11-9-2017

A Study of Very Short Intermittent DDoS Attacks on the Performance of Web Services in Clouds

Huasong Shan

A STUDY OF VERY SHORT INTERMITTENT DDOS ATTACKS ON THE PERFORMANCE OF WEB SERVICES IN CLOUDS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science and Engineering

by
Huasong Shan
B.S., Huazhong University of Science and Technology, 2003
M.S., Huazhong University of Science and Technology, 2006
December 2017

ACKNOWLEDGMENTS

This dissertation would not have been possible without the help and support of a large number of people.

First and foremost, I would like to thank my advisor, Prof. Qingyang Wang. I thank him for helping me to find a unique and fantastic research directory, very short intermittent DDoS attacks. Precisely due to his fundamental research, I can finish this dissertation during my PhD career so smoothly and rapidly. I will never forget every meeting and discussion with him during the last several years. His comprehensive knowledge and creative insight gives me a lot of motivations and guidances for my research. His encouragement and support helps me stand up to frustration of submitting the peer-review papers.

My deepest appreciation also extends to Prof. Seung-Jong Park and Prof. Gerald Baumgartner for taking their time to serve as my committee members. Numerous thanks go to them for allowing me to draw upon their experiences and technical knowledge, which have improved the quality of this research.

I would also like to thank Prof. Charles J. Monlezun for serving on my committee as Dean's Representative, even in Summer holiday. His suggestions and comments are highly appreciated.

There are too many colleagues and friends to mention all who have made my time at LSU memorable. I would like to thank all of them: Dr. Jian Tao, Fan Qi, Shungeng Zhang, Hui Chen, Shuai Yuan, Du jin, Xiangyu Lin, Chui-hui Chiu and all people who helped me during my PhD study.

Finally, I will be forever grateful to my parents, my wife Na Yang, my son Hongyi Shan. Without their support and help, this journey would have been impossible.

This research has been partially funded by National Science Foundation by CISE's CNS 1566443, Louisiana Board of Regents under grant LEQSF (2015-18)-RD-A-11.

TABLE OF CONTENTS

ABSTRACT

Distributed Denial-of-Service (DDoS) attacks for web applications such as e-commerce are increasing in size, scale, and frequency. The emerging elastic cloud computing cannot defend against ever-evolving new types of DDoS attacks, since they exploit various newly discovered network or system vulnerabilities even in the cloud platform, bypassing not only the state-of-the-art defense mechanisms but also the elasticity mechanisms of cloud computing.

In this dissertation, we focus on a new type of low-volume DDoS attack, Very Short Intermittent DDoS Attacks, which can hurt the performance of web applications deployed in the cloud via transiently saturating the critical bottleneck resource of the target systems by means of external attack HTTP requests outside the cloud or internal resource contention inside the cloud. We have explored external attacks by modeling the n-tier web applications with queuing network theory and implementing the attacking framework based-on feedback control theory. We have explored internal attacks by investigating and exploiting resource contention and performance interference to locate a target VM (virtual machine) and degrade its performance.

CHAPTER 1
INTRODUCTION

Distributed Denial-of-Service (DDoS) attacks for web applications such as e-commerce are increasing in size, scale and frequency [1, 2]. Akamai's "quarterly security reports Q4 2016" [1] shows the spotlight on Thanksgiving Attacks, the week of Thanksgiving (involving the three biggest online shopping holidays of the year: Thanksgiving, Black Friday, and Cyber Monday) is one of the busiest times of the year for the retailers in terms of sales and attack traffic. On the other battlefield, Berkeley's paper "above the clouds: a Berkeley view of cloud computing" [3] first introduced Cloud Computing in 2009, they predicted the top one opportunity for adoption of Cloud Computing is that it can use elasticity to defend against DDOS attacks. Until today, Cloud Computing has a rapid development and been widely adopted. A large number of web-sites are moved onto the cloud [4, 5]. However, DDoS attacks are still very active and even more severe, since ever-evolving new types of DDoS attacks that exploit various newly discovered network or system vulnerabilities even in the cloud, bypassing not only the state-of-the-art defense mechanisms [6, 7] but also the elasticity mechanisms of Cloud Computing.

Previous researches on performance bottleneck and interference in the cloud, demonstrate the unique features of performance issues in the cloud as follows: (i) the period of bottleneck is very short, especially within sub-second duration [8,9]; and (ii) the dependence among various resources is very strong, even cross different virtual/physical machines [10]; and (iii) resource contention and performance interference occurs when the utilization of shared hardware/software resources becomes high [11,12]; and (iv) the source of bottleneck is usually dynamic and unpredictable [13]; and (v) the adverse impact of application performance is very remarkable [14–16]. These new identified performance problems in the cloud motivated me to study the practicality and impact of the hypothetical attacks, especially for tail-latency sensitive use-facing Web applications, e.g., search engine, e-payment, online gaming.

In web applications such as e-commerce, fast response time is critical for service providers' business. For example, Amazon reported that an every 100ms increase in the page load is correlated to a decrease in sales by 1% [17]; Google requires 99 percentage of its queries to finish within 500ms [18]. In practice, the tail latency, rather than the average latency, is of particular concern for mission-critical web-facing applications [18–21]. Only requests with response time within the specified threshold have a positive impact to service providers' business, and the requests with long response time (beyond the threshold), not only waste network and system resources, but also cause penalties (negative impact in revenue) to the business of the service provider. In general, 99th, 98th, and 95th percentile response time are representative metrics to measure the performance of web applications [19, 22, 23]. In this work, we also use percentile response time (tail latency) as the evaluation metric to measure the effectiveness of the proposed attacks. The application scenario of the proposed attacks is focusing on Web applications, because they are not only most widely used services in the cloud, but also more sensitive on response time such as e-commerce websites.

Traditional Denial Of service (DoS) attacks hurt the availability of the target service by overloading its resources [6, 7, 24]. In the cloud environments, the attacking approaches are much more flexible and abundant, can be categorized in two classes: external and internal attacks [25, 26]. External attacks are the most orthodox form of DoS [6, 7, 24]. For external attacks, our hypothesis is that the external HTTP requests supported by the victim websites deployed in the cloud can cause sudden jump of resource demand flowing into the target system and temporally cause resource bottleneck in the weakest point of the target system, which in turn hurt the performance of service, such as very long response time perceived by the legitimate users. In contrast, internal attacks are new-born, which are emerging simultaneously with Cloud Computing by virtualization technique [25, 26]. For internal attacks, our hypothesis is that the adversarial programs in the co-located VMs [27] (on the same host with the target VM) can cause resource contention and performance

interference of the target VM and hurt its performance of service, the same as the severe consequences caused by external attacks [28, 29].

In this work we present a new low-volume application-layer DDoS attack called Very Short Intermittent DDoS (VSI-DDoS) attacks. A VSI-DDoS attacker creates intermittent performance interference bursts by carefully chosen legitimate external HTTP requests outside the cloud or internal resource contention inside the cloud, with the purpose of creating "Unsaturated DoS", where the denial of service can be successful for short periods of time (e.g., hundreds of milliseconds). VSI-DDoS attacks are not to bring down the system as traditional flooding DDoS attacks do, but rather to degrade the quality of service by causing frequent and sometimes intolerable delays for legitimate users (e.g., long-tail latency), which will eventually damage the long-term business goal of the target system. Through highlighting the practicality and impact of the proposed attacks, I hope that this work will draw attention of cloud providers to furnish much securer infrastructure to develop and deploy Web applications in the cloud.

1.1 Dissertation Statement and Contributions

My dissertation statement is formulated as follows: Very Short Intermittent DDoS Attacks expose significant challenges on tail latency of Web applications deployed in the cloud that can be effectively detected and mitigated through fine-grained time-correlation analysis.

To support the statement, we make the following three concrete contributions:

- We describe external and internal Very Short Intermittent DDoS (VSI-DDoS) Attacks in a stealthy manner that can broadly threaten a wide range of web applications deployed in the cloud. As for external attacks, the attacker sends intermittent pulses of legitimate HTTP requests to the target web system, causing frequent short-term denial of service by transiently saturating the bottleneck resource of the system (last within sub-second duration). As for internal attacks, we launch the internal attacks by exploiting the co-located VMs (the target VM is in the same host) as the attacking

VMs to trigger resource contention of non-partitionable shared hardware resource in the same host. This attack creates intermittent interference bursts of resource contention in the host of the cloud, causing frequent short-term denial of service by transiently saturating the bottleneck resource of the system. Since the average utilization of the target system under VSI-DDoS attacks is usually in moderate level (e.g., about 50-60%), tracing the cause of the performance problem is very difficult.

- We develop the attacking framework that is able to efficiently and adaptively launch the proposed attacks against a target web application. We model the impact of our attacks in n-tier systems based-on queue network models, which can effectively guide the attacks in an even stealthy way. And then we implement the attacking framework based-on the feedback control theory (e.g., Kalman filter) that allows our attacks to fit the dynamics of background requests or system state by dynamically tuning the optimal attack parameters. Through a representative web application benchmark (RUBBoS [30]) under realistic cloud scaling settings and equipped with the most popular state-of-the-art DDoS defense tools, we validate the practicality of the attack framework. We find that an appropriate combination of these attack parameters not only achieves high attacking efficiency, but also escapes the radar of the most popular state-of-the-art DDoS defense tools, even though the web application benchmark is deployed in the elastic public cloud platform (e.g., Amazon EC2, Microsoft Azure).

- We investigate potential practical approaches to detect or mitigate the proposed attacks. Due to the unique feature of the proposed attacks (very short-lived and intermittent), we exploit fine-grained monitoring (e.g., 50ms) to correlate the measured performance bottleneck with the potential attacking requests and detect the external attackers. To mitigate the internal attacks, the potential approach is to record the performance count in the host and reschedule the placement of the adversarial and victim VMs (e.g., live migration).

1.2 Organization of This Dissertation

We outline the remainder of this dissertation as follows. Chapter 2 describes external Very Short Intermittent DDoS attacks from high-level point of view. Chapter 3 future goes deep into external VSI-DDoS attacks, termed Tail attacks, modeling the attack impact and implementing a feedback control based attack framework. Chapter 4 extends VSI-DDoS attacks from external attacks outside the cloud to internal attacks inside the cloud. Chapter 5 concludes this work and discuss some future work.

CHAPTER 2
VERY SHORT INTERMITTENT DDOS ATTACKS

In this chapter [1] , we present a new class of low-volume application layer DDoS attack– Very Short Intermittent DDoS (VSI-DDoS). Such attack sends intermittent bursts (tens of milliseconds duration) of legitimate HTTP requests to the target website with the goal of degrading the quality of service of the system and damaging the long-term business of the service provider. VSI-DDoS attacks can be especially stealthy since they can significantly impair the target system performance while the average usage rate of all the system resources is at a moderate level, making it hard to pinpoint the root-cause of performance degradation. We develop a framework to effectively launch VSI-DDoS attacks, which includes three phases: the profiling phase in which appropriate HTTP requests are selected to launch the attack, the training phase in which a typical Service Level Agreement (e.g., 95^{th} percentile response time < 1 second) is used to train the attack parameters, and the attacking phase in which attacking scripts are generated and deployed to distributed bots to launch the actual attack. To evaluate such VSI-DDoS attacks, we conduct extensive experiments using a representative benchmark web application under realistic cloud scaling settings and equipped with some popular state-of-the-art IDS/IPS systems (e.g., Snort), and find that our VSI-DDoS attacks are able to effectively cause the long-tail latency problem of the benchmark website while escaping the radar of those DDoS defense tools.

2.1 Introduction

Distributed Denial-of-Service (DDoS) attacks for Internet services such as social networks and e-commerce are increasing in sophistication and scale. Kaspersky Lab's "DDoS Intelligence Report Q1 2017" [31] reports that the trend of DDoS attacks has been increasing despite numerous DDoS defense mechanisms. One important reason of the increasing

[1]Parts of this chapter have been previously published as: H. Shan, Q. Wang, and Q. Yan, "Very short intermittent ddos attacks in an unsaturated system," in Proceedings of the 13th International Conference on Security and Privacy in Communication Systems. Springer, 2017, Reprinted by permission.

popularity of DDoS attacks is due to the ever-evolving new types of DDoS attacks that exploit various newly discovered network or system vulnerabilities, bypassing the state-of-the-art defense mechanisms [6,7]. The damage that these DDoS attacks cause to enterprise organizations is well-known, and includes both monetary (e.g., $40,000 per hour) and customer trust losses [32]. Therefore, for guarding these Internet services, it is very important to detect, prevent and mitigate various emerging DDoS attacks.

In this work, we present a new low-volume application-layer DDoS attack called Very Short Intermittent DDoS (VSI-DDoS). A VSI-DDoS attacker sends intermittent bursts of carefully chosen legitimate HTTP requests to the target system, with the aim of creating "Unsaturated DoS", where the denial of service can successfully last for short periods of time (i.e., hundreds of milliseconds). VSI-DDoS attacks are not to bring down the system as traditional flooding DDoS attacks do, but rather to degrade the quality of service by causing frequent and sometimes intolerable delays for legitimate users, which will eventually damage the long-term business goal of the target system. For example, given that modern web applications care more about the tail latency than the average latency [19] (e.g., Google requires 99 percentage of its web-search to finish within 0.5s [18]), a long-tail latency (e.g., 95^{th} percentile response time $>$ 1s) caused by a VSI-DDoS attack can significantly affect the target website's business and reputation.

Compared to previous research on network-layer pulsating DDoS attacks [33–38], VSI-DDoS is a type of low-volume application-layer DDoS attacks, with even lower level of traceability and better stealthiness. Unlike the network-layer pulsating attacks which intend to temporarily saturate the bandwidth of network links that connect to the target system, VSI-DDoS attacks aim to create very short saturations of the bottleneck resource (usually in CPU or disk I/O) inside the target system, which we refer to as very short bottlenecks (VSBs) and typically require much less amount of attack traffic to trigger them. Less amount of attack traffic leads to a higher level of stealthiness. In addition, we adopt legitimate HTTP requests, which can easily penetrate the defense mechanisms adopted by

CDNs, network routers or switches in the path to the target system, thereby reducing the detection surface.

To effectively mount VSI-DDoS attacks, we should fully understand the triggering conditions of VSBs inside the target system, and quantify their long-term damages on the overall system performance. We develop a three-phase framework to tackle these challenges, which involves profiling, training, and attacking. Specifically, in the profiling phase we profile all the HTTP requests supported by the target website and select a set of appropriate ones to launch the attack. We find that heavy requests (e.g., the request with long service time consuming more bottleneck resource in the target web system) can achieve significantly better attack efficiency than light requests; only a small burst of heavy requests are needed to trigger VSBs of the target system, reducing the cost of an effective attack. In the training phase we use a typical Service Level Agreement (e.g., 95^{th} percentile response time < 1 second) for most e-commerce websites as an evaluation metric to train the key parameters of an effective VSI-DDoS attack, including burst volume, length, and interval. We find that an appropriate combination of these parameters not only achieves high attacking efficiency, but also escapes the radar of the most popular state-of-the-art DDoS defense tools, which further validates the stealthiness of the proposed attack.

In summary, the main two contributions of this work are:

- We present a novel low-volume application-layer VSI-DDoS attack that can broadly threaten a wide range of web applications in a stealthy manner. Unlike the traditional brute-force DoS attacks or pulsating attacks which focus on network bandwidth, VSI-DDoS attacks target the bottleneck resource of the target web system using legitimate HTTP requests, thereby reducing the cost of an effective attack while keeping the attack highly stealthy.

- We develop a three-phase framework via an empirical approach that is able to efficiently launch VSI-DDoS attacks against a target web application. Through a representative web application benchmark under realistic cloud scaling settings and

equipped with the most popular state-of-the-art DDoS defense tools, we validate the practicality of our attacking framework.

Through our evaluation of VSI-DDoS attacks under realistic cloud scaling settings and IDS/IPS systems, we confirm that the attacks not only bypass the triggering conditions of the cloud scaling but also invalidate capacity-based threshold monitoring and detection. We further explore two potential solutions to VSI-DDoS attacks: fine-grained VSBs detection and user behavior model validation, and discuss their strengths and weaknesses in practice.

The rest of this chapter is organized as follows. Section 2.2 presents the origin and motivation of VSI-DDoS attacks. Section 2.3 describes the definition of VSI-DDoS attacks. Section 2.4 presents the design of VSI-DDoS attack framework. Section 2.5 evaluates the effectiveness and stealthiness of our attacks. Section 2.6 discusses some countermeasures. Section 2.7 presents the related work and Section 2.8 concludes the chapter.

2.2 Background and Motivations

2.2.1 Origin of VSI-DDoS attacks

VSI-DDoS attacks originate from the new phenomenon of very short bottlenecks (VSBs), also called transient bottlenecks in recent performance analysis of Internet services deployed in Cloud environments [8,9]. In these previous studies VSBs have been identified as one of the main sources for the puzzling performance anomalies of the cloud-host web applications even though the system is far from saturation. From time to time cloud practitioners have reported that n-tier web applications produce very long response time (VLRT) requests on the order of several seconds, when the system average utilization is only about 50-60%. The VLRT requests themselves do not contain bugs, since the same requests return within tens of milliseconds when no bottleneck exists in the target system. The reason why VSBs can turn these normal short requests into VLRT requests is because VSBs can cause a large number of requests to queue in the system within a very short time. Due to some system level concurrency constrains (e.g., limited threads) of component servers, additional requests that exceed the concurrency limit of any component server will be dropped, caus-

ing TCP retransmissions (minimum time-out is 1 second [39]). The requests encountered TCP retransmissions become VLRT requests perceived by the end users.

While most of previous studies focus on VSBs caused by internal system level factors such as Java garbage collection, CPU Dynamic Voltage and Frequency Scaling (DVFS), interference of collocated virtual machines, this newly identified system vulnerability (VSBs) also motivates us to study the hypothetical VSI-DDoS attacks. Our hypothesis is that the external burst of legitimate HTTP requests can cause erratic fluctuation of resource consumption to be injected into the target system and cause VSBs in the weakest node of the whole distributed system, which in turn cause queue overflow and VLRT requests resulting from TCP retransmissions. Such VSI-DDoS attacks can potentially impose significant threats on current cyber infrastructures while remaining stealthy under the radar of state-of-the-art DDoS defense mechanisms and IDS/IPS systems.

2.2.2 Importance of Tail Latency

In web applications such as e-commerce, rapid responsiveness is vital for service providers' reputation and business. For example, Google requires 99 percentage of its web-search to finish within 0.5s [18]; Amazon reported that an every 100ms increase in the web-page load reduces sales by 1% [17]. In practice, the tail latency, instead of the average latency, is of special concern for mission-critical web-facing applications [18–21]. In shared infrastructures such as cloud environments, service level agreements (SLAs) are commonly used for specifying desirable response times, typically within one or two seconds [19]. In this case, only requests with response time within the specified threshold have a positive impact to service providers' business, and the requests with long response time (beyond the threshold), not only waste network and system resources, but also cause penalties (negative impact in revenue) to the business of the service provider. In general, 99th, 98th, and 95th percentile response time are representative metrics to measure the performance of web applications [19, 22, 23]. In this chapter, we also use percentile response time as the evaluation metric to measure the effectiveness of an adversary's VSI-DDoS attacks.

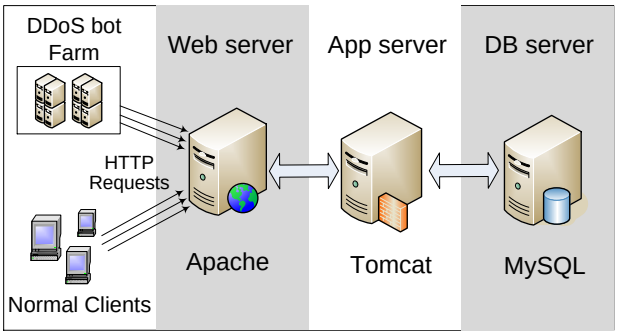


Figure 2.1. A 3-tier experimental sample topology. Appache as Web Server, Tomcat as Application Server, MySQL as Database Server.

2.2.3 Measured Long-Tail Latency Caused by VSI-DDoS Attacks

Here, we show the impact of VSI-DDoS attacks through concrete benchmark results. The benefit of benchmark experiments is to have a fully controlled system, which enables a detailed study about how the target system behaves when it is under a VSI-DDoS attack. The design of the VSI-DDoS attack framework and the real production setting evaluation are in Section 2.4 and Section 2.5, respectively.

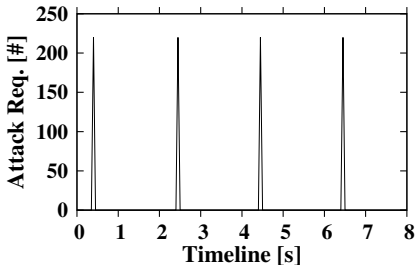
We use RUBBoS [30], a representative n-tier web application benchmark modeling the popular news forum website such as *Slashdot*. In Figure 2.1, we show the basic configuration for RUBBoS using the typical 3-tier architecture, with 1 Apache web server, 1 Tomcat application Server, and 1 MySQL database server deployed in an academic cloud platform (more details of experimental setup in Section 2.5.1). RUBBoS can emulate the behavior of legitimate users to surf the website. Each user follows a Markov chain model to navigate among different webpages, with averagely 7-second think time between receiving a web page and submitting a new page request. On the other hand, we adopt *Apache Bench* to send intermittent bursts of carefully chosen legitimate HTTP requests; each burst is injected within a very short time window (e.g., 50ms).

The mechanism of how VSI-DDoS attacks impact the performance of the target n-tier web system can only be seen using fine-grained monitoring. Figure 2.2 shows such an analysis when the target 3-tier benchmark website serving 3000 legitimate users is under

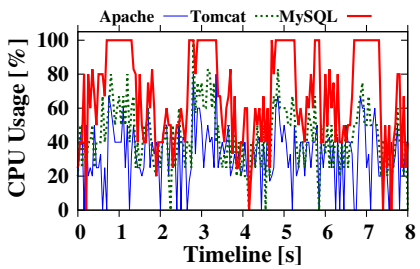
a VSI-DDoS attack. All the metrics in the subfigures are measured at every 50ms time window. Figure 2.2, a shows that the burst of attacking requests occurs in every 2 seconds. Each burst contains about 250 legitimate HTTP requests supported by the benchmark website within a 50ms time window. The bursts of attack requests cause transient CPU saturations of MySQL in Figure 2.2, b. These transient CPU saturations create VSBs and cause requests to queue in MySQL; MySQL local queue soon fills up (at 0.5s, 2.5s, 4.5s, and 6.5s), pushing requests to queue in upstream Tomcat and Apache in Figure 2.2, c. Once the queued requests in the front-most Apache exceed its queue limit (180 in our configuration), new requests from legitimate users will be dropped, leading to TCP retransmissions and very long response time (VLRT) requests as we observed in Figure 2.2, d.

We also find that the VSBs cannot be observed by our normal system monitoring tools (e.g., sar, vmstat, top) with the typical 1-second monitoring granularity. Figure 2.3, a shows that the CPU utilization of each server in the system is not saturated all the time using vmstat during the 8-second VSI-DDoS attacking period. Figure 2.3, b shows the outgoing/incoming network traffic of Apache is at very low rate ($<$ 10 MBps). We omit the graphs of resource utilization of other resources (e.g., memory, disk I/O) since all of them are far from saturation. Given such monitoring data, it is difficult for system administrators to trace the cause of the performance problem.

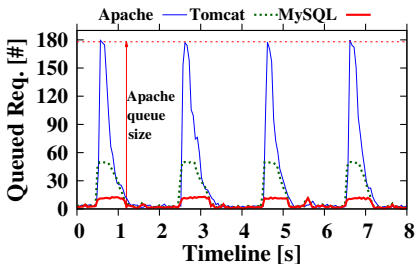
To illustrate the negative impact of the VSI-DDoS attack, we compare the percentile response time serving 3000 concurrent legitimate users by the target system under attack and without-attack in Figure 2.4. The percentile response time of the system under attack (the red line) uses the same dataset in Figure 2.2. This figure shows that all the requests finish within 200ms without attack (the black line). However, in the attacking scenario, the 95th percentile response time of the target system already exceeds 1 second, clearly showing the long-tail latency problem caused by the VSI-DDoS attack. Such long-tail latency problem is regarded as severe performance issue by most web applications, especially modern e-commerce (e.g., Amazon), as we have introduced in Section 2.2.2.



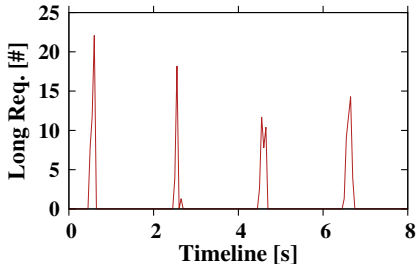
(a) Bursts of attacking HTTP requests in a VSI-DDoS attack during an 8-second time period. We count the # of attacking requests in every 50ms time window.



(b) Transient CPU saturations in MySQL coincide with bursts of attacking requests in Figure 2.2, a, suggesting that the VSI-DDoS attack creates frequent VSBs in MySQL.



(c) VSBs in MySQL (see Figure 2.2, b) cause requests to queue in MySQL, which in turn push requests to queue in Tomcat and Apache. The horizontal line shows the queue limit in the front-most Apache; once exceeded, new requests are dropped.

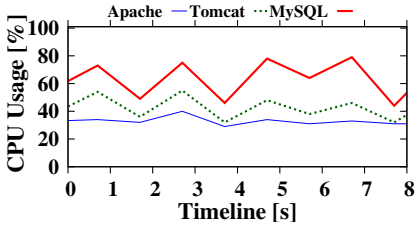


(d) Number of requests (from legitimate users) with response time $>$ 1s during the same 8-second time period. Such long requests are caused by TCP retransmissions once the front-most Apache drops incoming requests (see Figure 2.2, c).

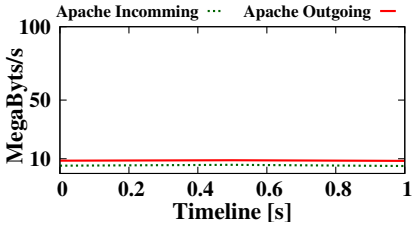
Figure 2.2. Measured performance of the benchmark application under a VSI-DDoS attack. Bursts of attacking HTTP requests (Figure 2.2, a) trigger VSBs in the bottom-most MySQL of the system (Figure 2.2, b), which cause requests to queue from local to the front-most Apache (Figure 2.2, c). Queue-overflows occur in Apache, causing TCP retransmissions and long response time requests (Figure 2.2, d).

2.3 VSI-DDoS Attack Overview

In a VSI-DDoS attack scenario the adversary is to create frequent VSBs in the target web system by sending intermittent bursts of legitimate HTTP requests to the target system without being detected. So the goal of VSI-DDoS attacks is not to bring the



(a) CPU util. of each server during the same 8s attacking period as in Figure 2.2.



(b) Incoming/outgoing network traffic of Apache. The bandwidth is 1 Gbps.

Figure 2.3. Resource utilization of each server in the system under the VSI-DDoS Attack. Metrics are measured using vmstat at every 1 second. Figure 2.3, a and 2.3, b show that both CPU and network bandwidth are not saturated. Utilization of other resources (e.g., memory) are omitted since they are far from saturation.

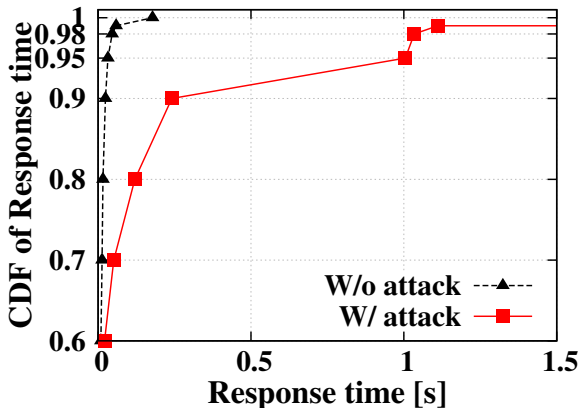


Figure 2.4. Percentile response time of the system serving 3000 legitimate users without and with the attack. The 95th percentile response time under attack is > 1 second, clearly showing the long-tail latency problem caused by the VSI-DDoS attack.

system down as traditional flooding DDoS attacks do, but rather to degrade the quality of service by causing frequent and sometimes intolerable delays for the legitimate users, which will eventually damage the business of the target system in the long run. Such attacks are stealthy because the target web system is in an "unsaturated state"; the duration of each created VSB is very short (e.g., 50ms), which can easily escape the detection of normal monitoring tools adopting coarse granularity statistical analysis (e.g., seconds or minutes).

To effectively launch a VSI-DDoS attack, we assume that all the bots under control are coordinated and synchronized so that requests generated by these bots can reach the

target web application at the same time or within a very short timeframe. This assumption is reasonable because many previous research efforts already provide solutions, using either centralized [33,40,41] or decentralized methods [35], to coordinate bots to send synchronized traffic and cause network congestion at a specific link. Our focus in this chapter is how to create frequent bursts of attacking but legitimate HTTP requests that can effectively trigger VSBs in the target system, causing long-tail latency of the target web system while avoiding being detected. We formally propose VSI-DDoS attacks as follows (Figure 2.5):

$$Effect = \mathbb{A}(V, L, I) \tag{2.1}$$

where,

- $Effect$ is the measure of attacking effectiveness; we use percentile response time as a metric to measure the tail latency of the target web system (e.g., 95th percentile response time $>$ 1s). $Effect$ is a function of V, L, I.
- V is the number (volume) of attack requests per burst. V should be large enough to temporarily saturate the bottleneck resource in the target system and trigger VSBs. At the same time, V should be small enough to bypass the state-of-the-art threshold-based detection tools [6, 7].
- L is the length of each burst. The total requests per burst V will be sent out during the period L. Thus the instant request rate to the target website during a burst period is V/L. L should be short enough to guarantee high instant request rate to trigger VSBs in the target system. Contrarily, too short L will cause large portion of attack requests dropped by the target system due to instant queue overflow (too high V/L), without causing any damage to the target system performance.
- I is the time interval between every two consecutive bursts. I infers the frequency of bursts of HTTP requests to the target system. I should be short enough so that the

attacker can generate bursts of HTTP request frequent enough to cause significant performance damage on the target system. On the other hand, too short I makes the attack similar to the traditional flooding DDoS attacks, which can be easily detected.

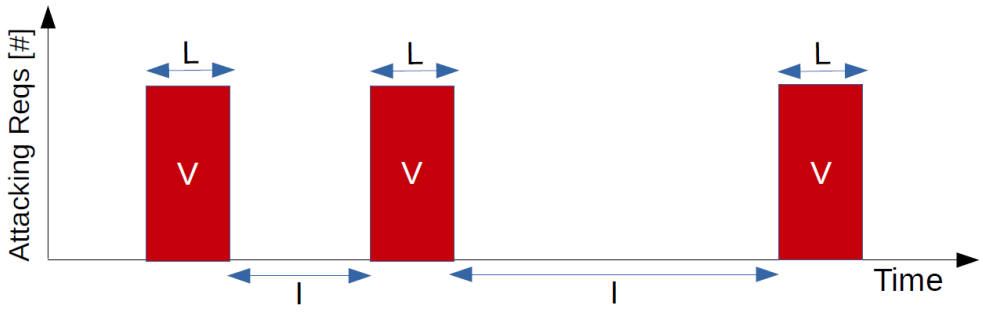


Figure 2.5. An illustration of a VSI-DDoS attack, which consists of burst volume V of HTTP requests, burst length L, and burst interval I.

We note that all the three components need to be carefully coordinated and tuned in order to launch an effective VSI-DDoS attack. To evaluate the effectiveness of such an attack, we measure the tail latency of the target website. We assume that the attack achieves its goal if the measured percentile response time under attack exceeds the predefined threshold, which depends on the SLAs of the target website. Based on this evaluation criteria, we develop an attacking framework which is able to estimate an optimal value of each parameter using an empirical approach for an effective VSI-DDoS attack in the following subsection.

2.4 VSI-DDoS Attack Framework

To launch effective VSI-DDoS attacks, we present an attacking framework. The proposed VSI-DDoS attack framework contains three phases: profiling, training, and attacking (Figure 2.6). The profiling phase gets attack requests by profiling the target web system. The training phase is to determine the three parameters of a VSI-DDoS attack. The attacking phase generates and deploys attacking scripts to distributed bots and launches the actual attack.

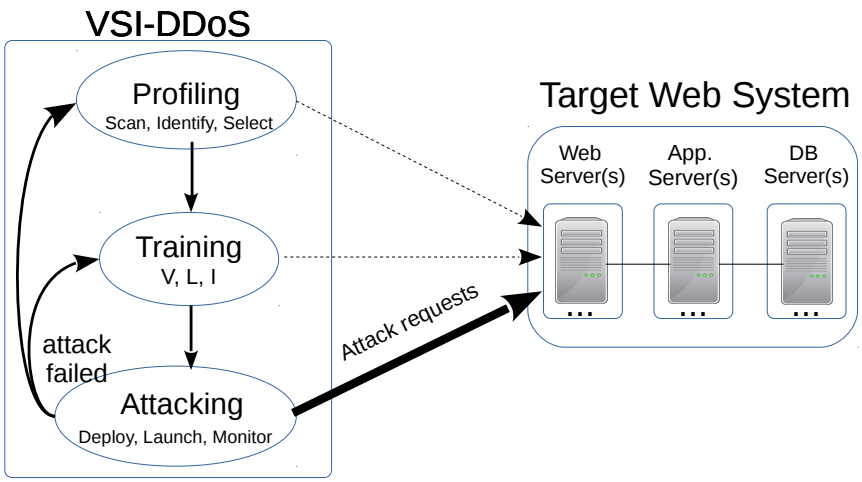


Figure 2.6. VSI-DDoS Attacks framework involving three phases: profiling, training, and attacking.

2.4.1 Profiling phase

This phase selects appropriate types of HTTP requests for attacking in order to create VSBs in the target website with the minimum cost, meaning the least number of attacking requests for each burst. This phase includes the following three steps.

(1) Scanning the supported HTTP requests. To select appropriate HTTP requests, one challenge is to retrieve representative HTTP requests that cover all the transaction types supported by the target website, including both static and GET/POST dynamic requests. Although the requests for static content of a website are easy to retrieve by using some crawling tools (e.g., scrapy), requests for dynamic content are more difficult to get. This is because POST requests are sent out by submitting forms (content is in the body section of a HTTP request), not through the direct URLs. To solve this issue, we adopt a script-based open source web browser PhantomJS to retrieve and analyze the form tags inside the HTTP response for every HTTP request. After the attacker provides some initial values for associated input boxes (e.g., user-name and password) inside each form, PhantomJS can submit POST requests automatically. PhantomJS also supports cookies which allows an attacker to conduct consecutive interactions with some websites (e.g., Facebook) which require user login before further website navigation. POST requests are

important types of attack requests because they can penetrate Content Delivery Networks (CDN) and attack the original target website. CDNs are widely used by websites nowadays to improve the website performance by caching static content. Since POST requests are dynamic requests which typically require to retrieve/write dynamic information from/to the back-end database, current CDN vendors usually do not support caching responses for POST requests [42]. Thus POST requests are natural candidates to launch effective VSI-DDoS attacks for websites with CDN support.

(2) Identifying heavy requests using service time. Once we get enough supported HTTP requests, the next challenge is to decide which requests consume more bottleneck resource (e.g., Database CPU) of the target web system than the others. We term the requests heavily consuming the bottleneck resource as heavy requests (e.g., POST requests), meanwhile, those consuming no or little bottleneck resource as light requests (e.g., static requests). In this case, heavy requests are natural candidates to launch a VSI-DDoS attack because a fewer number of them are needed to trigger VSBs in the target web system than that of light requests. A low number of attacking requests per burst also make the attack stealthy because of the low volume of network traffic. The key question is how do we determine which requests are heavy and which are light? We use the service time of each type of HTTP requests as a key metric to distinguish the heavy requests from the light ones. Service time of a HTTP request is the time serving the request by the target web system without any queuing delay. Previous research results [9] show that the predominant part of the service time of a request is spent on the bottleneck resource in the system. When the target system is at low utilization (Low utilization is to rule out the queueing effect inside the target system), service time can be estimated to be the end-to-end response time of a request subtracting the network latency between the client and the target web application. The end-to-end response time of a HTTP request can be easily recorded using Apache Bench or PhantomJS. Lots of tools can be used to measure the network latency (e.g., the ping command). We measure the service time of each type of HTTP requests

multiple times and employ the average in order to mitigate influence of network latency variation.

(3) Selecting candidate requests. A naive strategy to select candidate attacking requests is always choosing the heaviest type of requests. In this case, the attacker can use the minimum number of requests per burst to create VSBs in the target web system. However, single type of attacking requests in a VSI-DDoS attack has the risk to be identified as abnormal by existing DDoS defense tools using statistical analysis. For example, the defense tool may simply aggregate all the requests sent out from the same IP and identify that some IPs only send one type of HTTP requests, which is highly suspicious. To bypass such statistics-based detection mechanisms [43], an attacker can select a set of top-ranked heavy requests, which can achieve the same attacking goal with slightly increased cost. Some more advanced defense mechanisms use machine-learning based techniques to learn a legitimate user behavior model [44] and infer suspicions requests if the sequence or the transition probability among them significantly deviates from the model (judging based on pre-defined thresholds). In this case the attacker needs to select candidate requests more carefully to make sure that the sequence of HTTP requests sent from a bot is feasible for a legitimate user. We will discuss this in more detail in Section 2.6.

2.4.2 Training phase

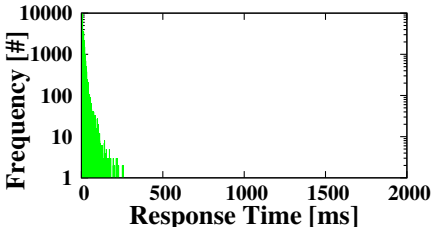
This phase is to train the key parameters (V,L, and I) of an effective VSI-DDoS attack that meets the adversary's goal (e.g., 95th percentile response time $>$ 1s).

(1) Training volume. The technical objective of a VSI-DDoS attack is to create frequent VSBs in the target web system. Thus a key challenge of VSI-DDoS attacks is to determine whether a batch of attacking requests are able to create a VSB in the target system or not. In most cases the attacker has no privilege to monitor the resource utilization of the target system. Thus the attacker cannot depend on internal resource monitoring to determine the occurrence of VSBs. However, we know that the occurrence of a VSB will create temporary request congestion inside the target system; once the queued requests exceed any system-

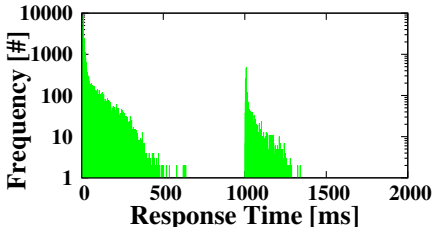
level queue capacity (e.g., thread pool size), new arriving requests will be dropped and TCP retransmissions (minimum time-out is 1 second) will happen, leading to long response time perceived by the end users (see Figure 2.2). In this case, long response time caused by queue-overflow and TCP retransmissions can be treated as a signal of the occurrence of VSBs. Given such an idea, we can gradually increase the volume of the attacking requests per batch until the observation of requests with abnormally long response time.

The minimum volume per burst that triggers the long response time requests due to TCP retransmissions is the lower bound V_{min} for the selected attack requests. Figure 2.7 shows the process of training V_{min} for an effective VSI-DDoS attack for our RUBBoS website, which is serving 3000 legitimate users. When burst volume is only 20, the response time perceived by all legitimate users is lower than 250ms in Figure 2.7, a. We increase burst volume step by step (e.g., 10 or 20) until we observe the requests sent from legitimate users experience abnormally long response time in Figure 2.7, b. The distinct two-modal response time distribution indicates that 100 reaches the lower bound. In practice we set the volume higher than V_{min} to guarantee the successful triggering of VSBs in the target system and achieve better attack result as shown in Figure 2.8. On the other hand, the volume should not be too high otherwise it will trigger the alarm of defense tools (e.g., Snort [43]) deployed in the target web system. We can increase the number of bots and reduce the number of attack requests per bot to bypass the state-of-the-art detection mechanisms.

(2) Training burst length. A good burst length L should maximize the impact of the burst of attacking requests on the requests sent from legitimate users. We observed that the best L should be the service time of the selected attacking requests. A HTTP request that originates from a client arrives at the web server, which distributes it among the application servers, which in turn ask the database to execute the query. Due to the RPC-style synchronous communication between consecutive tiers, the processing threads and other associate soft resources such as database connections of a component server will be occupied until all the activities in the downstream tiers are done. In order to create the



(a) 20 requests per burst case. The low response times of all the requests indicate no TCP retransmission.



(b) 100 requests per burst case. The observed response time over 1s is a signal of the occurrence of TCP retransmissions.

Figure 2.7. Training the lower bound of volume for an effective VSI-DDoS attack for our RUBBoS benchmark application. We increase the volume of attacking requests per burst step by step until we observe abnormally long response time of requests sent from legitimate users (see Figure 2.7, b).

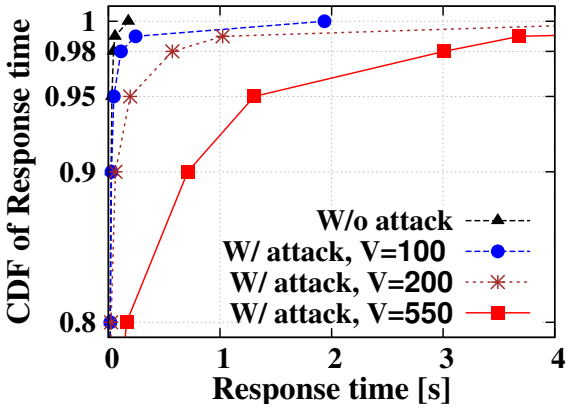


Figure 2.8. Impact of volume V per burst. This figure shows the percentile response time of the target system (serving 3000 legitimate clients) gets more damages as the attack volume per burst V increases from 100 to 550 (fixing L=50ms, I=7s).

most soft resource consumption, the burst of attacking requests should arrive within the service time of the attacking requests. In this case, all the attacking requests will stay in the target system before any of them finishes processing and leaves the system. Once any of soft resources in any tier of the system are exhausted, new requests from legitimate users will be dropped, leading to long response time requests resulting from TCP retransmissions.

Figure 2.9 shows the impact of the burst length L on the tail latency of our RUBBoS website. We choose the heavy request "ViewStory" as the attacking requests. The service

time of such heavy request is about 50ms. Note that the biggest attacking damage appears when L is 50ms. Too short L leads to low effectiveness for the attacking burst, since most of the attacking requests will be dropped due to sudden queue-overflow (Short L leads to high instant request rate V/L, OS kernel may not be able to handle packets promptly due to high overhead of interrupt handling [38]), without causing any harm to the requests from legitimate users. On the other hand, too long L (e.g., the L=800ms case) leads to low instant request rate (V/L), which may not be able to create VSBs in the target system and lead to inferior attacking results.

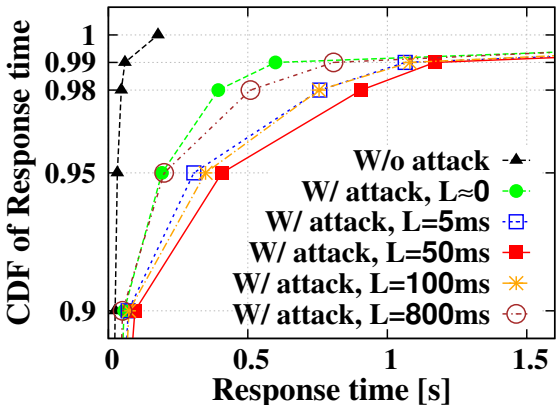


Figure 2.9. Impact of the length L per burst. The biggest attacking damage (serving 3000 legitimate clients, and fixing V=200, I=7s) appears when L is 50ms; the more the burst length deviates from 50ms, the weaker the damage caused by the VSI-DDoS attack is.

(3) Training interval between bursts. By determining V and L of each burst we can make sure one burst is able to trigger a VSB in the target system. The final goal of a VSI-DDoS attack is to create the long-tail latency problem of the target web system. Too small interval between bursts makes it similar as the traditional flooding DDoS attack, thus can be easily detected. Too large interval creates insufficient number of VSBs in the target web system, thus unable to achieve the adversary's goal. To select a reasonable interval, we start from a relatively large interval and gradually reduce the interval until the measured tail latency meets the adversary's goal. Figure 2.10 shows such a process of selecting a reasonable interval for our RUBBoS benchmark. To avoid an obvious burst

pattern of attacking requests, the interval between consecutive bursts is not necessarily assigned with a fixed value. A VSI-DDoS attacker can design the interval with a random variable following certain statistical distributions, with the mean to be similar to a normal user's think time between consecutive requests. Figure 2.2 in Section 2.2.3 is such an example. The interval between attack bursts follows a normal distribution with the mean of 2 seconds.

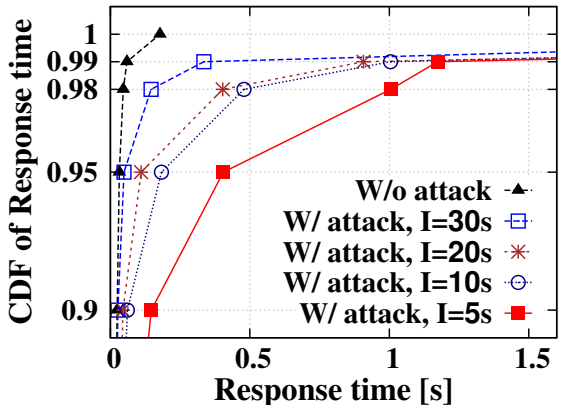


Figure 2.10. Impact of the interval I between bursts. This figure shows the percentile response time of the system (serving 3000 legitimate clients) gets more damages as we decreases I from 30s to 5s (fixing V=200, L=50ms).

2.4.3 Attacking Phase

This phase is to launch the real VSI-DDoS attack based on the previous profiling and training results of V, L, I for request bursts. Attackers launch the attack by leveraging the resources of multiple hosts, especially Botnet. We note that an attacker should use a probe to continuously monitor the performance of the target website, for example, send a sequence of very light HTTP requests (e.g., html) at regular intervals and check the response time distribution. The profiling and training phase need to redo once the attacking results cannot meet the adversary's goal due to the change of baseline workload or system state (e.g., dataset size change).

2.5 Evaluation

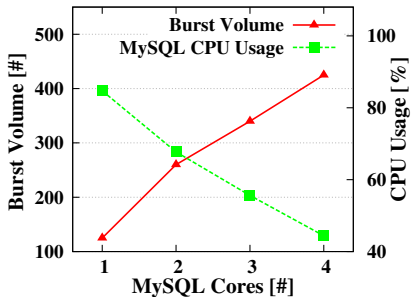
2.5.1 VSI-DDoS Attacks under Cloud Scaling

To evaluate the effectiveness of our VSI-DDoS attacks in the real production settings, we deploy RUBBoS in a popular NSF sponsored cloud platform-Cloudlab [45].

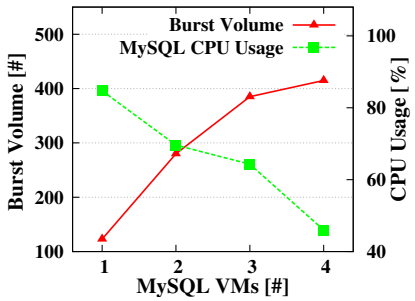
In the real production environment, once administrators pinpoint the performance bottleneck of an n-tier system, they can solve the issue by scaling the bottleneck tier. One policy is scaling up (updating the hardware of the bottleneck tier), and the other is scaling out (adding more machines/virtual machines to the bottleneck tier). For example, Amazon Auto Scaling [46] can scale out EC2 instances as the demand of an application increases. We evaluate our attack under both scaling settings. In our experiments, we assume that the bottleneck is MySQL since the bottleneck typically takes place in the database due to the high resource consumption of database operations. To evaluate our attack under the cloud scaling settings, we keep all the software configuration (e.g., queue size, DB connection pool size) the same to rule out their impacts to our evaluation. In the scaling up case, we update the hardware unit (1 CPU core and 1GB Memory) of MySQL from 1 to 4 Units. In the scaling out case, we increase the number of MySQL VMs from 1 to 4. All the VMs (Xen-based Emulab virtual nodes) are running CentOS 6.5 on 2.10GHz Intel Xeon E5-2450 processors. To maximize the impact of our attack, we set the burst length as the service time of attack requests (e.g., 50ms.), and set the burst interval as 2s. We conduct the attack experiments for 10 minutes in each scenario since half of the DDoS attacks last longer than 10 minutes [47]. Our DDoS bot farm in Figure 2.1 consists of 8 machines, one serves as a centralized controller that coordinates and synchronizes the other nodes to launch the attacks.

Figure 2.11 depicts the required burst volume and the relevant usage of the bottleneck resource at scaling up/out scenarios to achieve our attacking goal. We can see the average CPU utilization of MySQL reduces from high load ($> 80\%$) to moderate level ($< 50\%$) after the bottleneck tier is scaled, indicating the scaling policies are effective since more CPU

cores or VMs can mitigate the impact of the bottleneck resource to the system performance. However, we can still reach our attacking goal by increasing attacking volume per burst even in a large scale scenario, since it requires higher burst volume to trigger VSBs in the system after the capacity of the bottleneck tier increases. On the other hand, increasing the attack requests by each bot obviously increases the risk of detection by the target system, but we can coordinate more synchronized bots to send higher volume per burst to achieve our attacking goal using decentralized synchronization mechanism [35]. As such, we can still keep our attacks under the radar of the state-of-the-art detection mechanism.



(a) Scaling CPU cores of MySQL.



(b) Scaling Virtual Machines of MySQL.

Figure 2.11. The required burst volume and the CPU utilization of MySQL in scaling up/out scenarios while achieving our victim goal. Average CPU Usage decreases after scaled, indicating the scaling mitigates the bottleneck. However, we can still get the attacking goal by increasing burst volume.

In real production clouds (e.g., Amazon AWS), the users can customize some triggering conditions to instruct Amazon Auto Scaling [46] to scale out/in instances in response to metrics (such as bandwidth usage or CPU utilization) monitored by Amazon CloudWatch [48]. The monitoring granularity of Amazon CloudWatch for premium users is 1 minute [48]. For example, the target system can add more VMs once the average CPU utilization of all the instances exceeds 85% during a 1-minute statistical period. From Sampling Theory, our attack is hard to trigger the scaling plans sampling in minutes level, since the VSBs usually occur in milliseconds level (more details about monitoring granularity in Section 2.6). Large-scale web applications typically adopt dynamic scaling strategy for

better resource efficiency and load balancing, VSI-DDoS attacks can avoid the triggering conditions of the cloud scaling, thus we expect that the current cloud scaling techniques do not help resolving our attacks.

2.5.2 VSI-DDoS Attacks under Defense Tools

To validate the stealthiness of our attacks under the state-of-the-art defense mechanisms, we deploy some popular defense tools before the web tier (the role like a firewall) in our RUBBoS environments.

Snort is the most widely deployed network anomaly detection system in the world that is acquired by Cisco Systems on October 7, 2013, and widely used in practice for DDoS defense [43,49]. Snort.AD [50], extended based on Snort, is a threshold and statistics-based network anomaly detection tool, which can analyze the network traffic based on different protocols (UDP, TCP, HTTP, etc.) within a certain period. Here, we take HTTP traffic as a representative metric in Snort.AD to evaluate whether our attacks break through the cordon, since our attacking requests only involve the HTTP packets. In the following experiments, we configure 2000 and 4000 concurrent legitimate users as the baseline for low and high background workload scenarios. We set 95th, 98th and 99th percentile response time (>1s) as the candidate attacking goals, and call them 95th, 98th, and 99th case hereinafter. To achieve these different attacking goals, we fix the burst length as the service time of the attacking requests and the burst interval as 2s, and only tune the burst volume ((250, 150, 100) and (150, 100, 50) for the 95th, 98th and 99th case of 2000 and 4000 baseline, respectively). We conduct the experiments in a 10-minute period for each scenario. We modify the code of Snort.AD to trace the number of the HTTP incoming/outgoing packets in a minute interval and the HTTP incoming/outgoing speed in terms of Mega Bytes per second, to evaluate whether they exceed the threshold for these cases with different attacking goals and background workload.

How to set the alert threshold is a well-known challenge for administrators [51, 52]: a high threshold may not be able to detect anomalies; a low threshold may incur a high

number of false positive alarms which an administer tries to avoid in practice. Typically, a widely-adopted setting strategy [51,52] is to set the threshold of each monitoring metric based on the capacity of the target system. Network security company [51] recommends that the company's IT team should conduct the necessary performance tests to determine the capacity, and set the threshold lower than the capacity to prevent resource exhaustion (e.g., define the threshold when reaching 85% bottleneck resource utilization of the web system). We also profile the capacity of the target system in our experimental environment under a worst-case scenario (when serving 6000 concurrent users, the bottleneck resource, MySQL CPU utilization, reaches 85%). In our experiments, we take this widely-adopted strategy to define the alert threshold listed in Column 2 of Table 2.1 to capture our attacks.

Table 2.1 lists the maximal HTTP incoming/outgoing packets and speed under our attacks for the cases with different attacking goals and background workload. All of the measured traffic metrics are in the moderate level and far from the predefined threshold (based on system capacity) when the corresponding attacking goal is achieved, indicating that our attacks create an "Unsaturated illusion" for Snort. As a result, Snort reports no alert. More importantly, the increased traffic due to our attacks is small compared to the baseline case, especially when the attacking goal is less aggressive (e.g., 99th percentile response time $>$ 1s). For example, an effective VSI-DDoS attack in the 99th case only incurs 10% more traffic when the baseline workload is 2000, and 4% more traffic when the baseline is 4000. This result also suggests that an effective VSI-DDoS attack is easier to achieve as the background traffic increases.

The fundamental reason that our attack (in *milliseconds* level) can invalidate the traditional threshold-based detection tools (in *seconds or minutes* level) is their coarse monitoring granularity. The coarse monitoring granularity is effective for identifying brute-force DDoS attacks and flash crowds lasting for tens of seconds or minutes [53] (detailed explanation in Section 2.7), but obviously too long to observe any abnormal behaviors triggered by a VSI-DDoS attack lasting for only tens of milliseconds (e.g., the minimum measured

Table 2.1. Measured HTTP traffic in the cases of 95th, 98th and 99th percentile response time (>1s) as candidate attacking goals. All of measured metrics are less than the predefined thresholds set based on system capacity when the corresponding attacking goal is achieved. In.: HTTP Incoming, Out.: HTTP Outgoing, B/L: Baseline, Unit of packets: #/min, Unit of speed: MB/sec.

		2000 low load				4000 high load			
Metrics	Threshold	95th	98th	99th	B/L	95th	98th	99th	B/L
In. packets	299K	158K	119K	111K	99K	224K	214K	208K	201K
Out. packets	349K	171K	134K	127K	116K	259K	249K	241K	233K
In. speed	9.32	4.68	3.96	3.62	3.11	7.08	6.76	6.45	6.23
Out. speed	17.83	7.62	6.83	6.48	5.94	12.78	12.46	12.12	11.89

rate-interval of the Cisco Adaptive Security Appliance is 1 second [54], the minimum sampling interval of Snort.AD is 1 minute [50], the sampling interval of BotSniffer's monitor engine [55] is in seconds level). Indeed, fine-grained monitoring could mitigate the problem, but with the cost of high monitoring overhead and potentially high false positive alarms (falsely block legitimate users), because web application workload is naturally bursty [53]. We will discuss the impact of monitoring granularity in more detail in the following.

2.6 Discussion of Possible Countermeasures

Here, we introduce two more candidate countermeasures for VSI-DDoS attacks and discuss their pros and cons in practice.

2.6.1 Fine-Grained VSBs Detection

A natural way to detect a VSI-DDoS attack is to detect the occurrence of VSBs in the target web system, and determine whether they are caused by bursts of malicious HTTP requests. However, detecting VSBs in the target web system is challenging because they usually occur in milliseconds level; from Sampling Theory, these VSBs would not be reliably detectable by normal tools sampling at time intervals from 1 second (e.g., Snort, BotSniffer [55], sar, vmstat, top) to several minutes (e.g, CloudWatch). To reliably detect VSBs and their correlation with a potential VSI-DDoS attack, we need both the system and application level fine-grained monitoring (millisecond level). System-level monitoring is to detect VSBs by collecting the hardware resource utilization of all component servers

in the target system using fine-grained monitoring tools (e.g., collectl). Application level monitoring is to collect the request processing logs of each component server in the system and analyze the performance metrics such as incoming request rate, queue status, and point-in-time response time in fine granularity. Given the collected fine-grained monitoring data, we apply a timeline correlation analysis to link the observed VSBs in system-level monitoring with the application level performance metrics, as we have illustrated in Section 2.2.3 (see Figure 2.2). On the other hand, with "coarse" monitoring granularity (e.g., 1s), these metrics only show moderate variations or non-saturation (see Figure 2.3) over time, which will likely not bring any attention to administrators.

Although the fine-grained monitoring approach is conceptually simple, it requires sophisticated fine-grained monitoring tools. [8] shows that VSBs can be caused by the temporary saturation of any system resource that is in the execution path as HTTP requests flow via the system. Specifically, VSBs caused by a VSI-DDoS attack may not necessarily be in hardware resources, but in system soft resources (e.g., database locks, thread pool) that are out of the scope of existing fine-grained monitoring tools (e.g. collectl). We observed this phenomenon when we deploy Opentaps, a popular open source ERP/CRM web application, in Amazon EC2 cloud platform. The target Opentaps web application shows a significant long-tail latency problem under a VSI-DDoS attack while collectl reports no saturation of any hardware resources. In addition, monitoring overhead is another big concern of the fine-grained monitoring approach. In our RUBBoS experiments, we observe that collectl incurs high overhead at sub-second sampling intervals (about 6% CPU utilization overhead at 100ms interval and 12% at 20ms). How to balance the monitoring overhead and detection accuracy of a VSI-DDoS attack remains a significant challenge and future work.

2.6.2 User Behavior Model Validation

Some advanced defense mechanisms use machine-learning based techniques to learn a normal user behavior model from web server logs. These user behavior models [44, 56] are

used to differentiate HTTP requests sent by humans from those sent by bots. For example, Oikonomou et al. [44] model three aspects of human behaviors such as inter-arrival time of consecutive requests from the same user, choice of content to access, and ability to ignore invisible content. Such user behavior models are indeed effective if a VSI-DDoS attacker chooses single type of requests to attack or a set of heavy requests that have low transition probability among them. However, the attacker can set the interval between consecutive bursts in a VSI-DDoS attack similar to the browsing behavior of a legitimate user (e.g., an average 7-second think time between two webpages). The attacker can also learn a popular sequence of HTTP requests that a legitimate user will likely go through when visiting the target website. Then the same sequence of requests can be selected as the attack requests. Such selection strategy may not be the most cost-efficient one since not all of the selected requests are heavy, but the attacker can still achieve the goal by controlling more bots to launch the attack. Defense mechanisms that adopt user behavior models certainly raise the bar of our attacks, but they do not suffice to detect and defend such attacks. Future work can evaluate our attacks under more complicated machine-learning based mechanisms [44, 56].

2.7 Related Work

DDoS attack and defense mechanisms have been extensively investigated and categorized in survey papers [6, 7]. In this section, we review the most relevant work in two aspects: the low-rate network-layer pulsating DDoS attacks and the low-volume application-layer DDoS attacks [24].

Many attack mechanisms in this category [33–38] have been proposed, which send bursts of TCP packets to cause packet drops, by exploiting deficiencies in TCP retransmission time-out mechanism known as Shrew attack [37], congestion control response mechanism known as Pulsating attack [35, 36, 38], or the transients of the system's adaptation mechanisms known as RoQ attack [33]. These attacks share some similar features with our VSI-DDoS attacks, such as the low attacking volume in Shrew and Pulsating attack,

and the QoS degradation in RoQ attack. However, our work differs from these work in three aspects: (1) all these attacks are network-layer attacks targeting at network links, while our attacks are at the application-layer, exploiting the bottleneck resource (e.g., CPU, I/O) and the complex resource dependencies (e.g., push-back wave [8]) inside the web system; (2) these attacks usually require a fixed or crafted burst interval to synchronize the Retransmission Timeout (RTO) duration, while our attacks are more flexible in selecting burst volume, length and interval, which allows our attack to be even stealthier; (3) our evaluation metric is based on percentile response time, representing real user experience and provider's service level agreements, which has not been used previously to quantify the attack impact.

One class of low-volume application-layer DDoS attacks specifically related to our attacks are called flash crowds [53], which refer to the scenario when thousands of legitimate users intensively browse an e-commercial website due to a hot event (e.g., Black Friday deals). Previous detection mechanisms are mainly focusing on differentiating traffic from flash crowds created by legitimate users or application-layer DDoS attacks [53,56,57], such as using hidden semi-Markov model [56] and session-level misbehaviors [57] for anomaly detection. VSI-DDoS attacks can be launched with randomized interval and learn from the user behaviors of a legitimate user, which invalidates those user behavior model-based application layer detection mechanisms. More importantly, VSI-DDoS attacks exploit very short bottlenecks (VSBs) as the system vulnerability, VSBs can be much shorter (tens of milliseconds) than the duration of the traditional application-layer flash crowds traffic (tens of seconds or minutes). Thus, the detection mechanisms of identifying DDoS attacks from flash crowds can be defeated by our VSI-DDoS attacks.

2.8 Conclusion

We presented a new type of low-volume application layer DDoS attack, VSI-DDoS attacks, exploiting a newly discovered system vulnerability (VSBs) of n-tier web applications. Using concrete experimental results we showed that VSI-DDoS attacks can be specially ef-

fective and stealthy because they can cause an intolerable long-tail latency issue of the target system while the average usage rate of all the system resources is at a moderate level (Section 2.2.3). We developed a VSI-DDoS attacking framework in which an attacker can systematically profile the target web application and train key attacking parameters for an effective VSI-DDoS attack (Section 2.4). Through a representative web application benchmark under realistic cloud scaling settings and equipped with the most popular state-of-the-art DDoS defense tools, we validated the negative impact and stealthiness of VSI-DDoS attacks, and confirmed the practicality of our attacking framework (Section 2.5). We further explored the pros and cons of two possible countermeasures for our attacks (Section 2.6). VSI-DDoS attack, as a newfound DDoS attack, is an important contribution to complement emerging DDoS attacks.

CHAPTER 3
TAIL ATTACKS ON WEB APPLICATIONS

In this chapter [1], we describe a new type of low-volume application layer DDoS attack– Tail Attacks on Web Applications. Such attack exploits a newly identified system vulnerability of n-tier web applications (millibottlenecks with sub-second duration and resource contention with strong dependencies among distributed nodes) with the goal of causing the long-tail latency problem of the target web application (e.g., 95th percentile response time $>$ 1 second) and damaging the long-term business of the service provider, while all the system resources are far from saturation, making it difficult to trace the cause of performance degradation.

We present a modified queue network model to analyze the impact of our attacks in n-tier architecture systems, and numerically solve the optimal attack parameters. We adopt a feed-back control-theoretic (e.g., Kalman filter) framework that allows attackers to fit the dynamics of background requests or system state by dynamically adjusting attack parameters. To evaluate the practicality of such attacks, we conduct extensive validation through not only analytical, numerical, and simulation results but also real cloud production setting experiments via benchmark web application equipped with state-of-the-art DDoS defense tools. We further proposed a solution to detect and defense the proposed attacks, involving three stages: fine-grained monitoring, identifying bursts, and blocking bots.

3.1 Introduction

Distributed Denial-of-Service (DDoS) attacks for web applications such as e-commerce are increasing in size, scale and frequency [1, 2]. Akamai's "quarterly security reports Q4 2016" [1] shows the spotlight on Thanksgiving Attacks, the week of Thanksgiving (involving the three biggest online shopping holidays of the year: Thanksgiving, Black Friday, and Cyber Monday) is one of the busiest times of the year for the retailers in terms of

[1]Parts of this chapter have been previously published as: H. Shan, Q. Wang, and C. Pu, "Tail attacks on web applications," in Proceedings of the 24nd ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017, Reprinted by permission.

sales and attack traffic. Web applications remain the most vulnerable entrance for any enterprise and organization, so the attackers can exploit them to launch both low-volume and stealthy application-layer DDoS attacks. On the other hand, in web applications especially e-commerce websites, fast response time is critical for service providers' business. For example, Amazon reported that an every 100ms increase in the page load is correlated to a decrease in sales by 1% [58]; Google requires 99 percentage of its queries to finish within 500ms [18]. Emerging augmented-reality devices (e.g., Google Glass) need the associated web applications with even greater responsiveness (e.g., within 100 milliseconds) in order to guarantee smooth and natural interactivity. In practice, the tail latency, rather than the average latency, is of particular concern for response-time sensitive web-facing applications [18–21, 59].

In this chapter we present a new low-volume application-layer DDoS attack–Tail Attacks, significantly worsening the tail latency on web applications. Web applications typically adopt n-tier architecture in which presentation (e.g., Apache), application processing (e.g., Tomcat), and data management (e.g., MySQL) are physically separated among distributed nodes. Previous research on performance bottlenecks in n-tier systems [8–10] shows that very short bottlenecks (VSBs) or millibottlenecks (with sub-second duration) with dependencies among distributed nodes not only cause queuing delay in local tier, but also cause significant queuing delay in upstream tiers in the invocation chain, which will eventually cause the long-tail latency problem of the target system (e.g., 95th percentile response time longer than 1 second). More importantly, this phenomenon usually starts to appear under moderate average resource utilization (e.g., 50% or 60%) of all participating nodes, making it difficult to trace the cause of performance degradation. In the scenario of Tail Attacks, an attacker sends intermittent bursts of legitimate HTTP requests to the target web system, with the purpose of triggering millibottlenecks and cross-tier queue overflow, creating "Unsaturated DoS" and the long-tail latency problem, where denial of service can be successful for short periods of time (usually tens or hundreds of millisec-

onds), which will eventually damage the target website's reputation and business in the long term.

The study of Tail Attacks complements previous research on low-rate network-layer DDoS attacks [33–38, 60], low-volume application-layer DoS Attacks [61, 62], and flash crowds (usually tens of seconds or minutes) [53, 56, 57] which refer to the situation when thousands of legitimate users suddenly start to visit a website during tens of seconds or minutes due to a flash event (e.g., during the week of Thanksgiving). The uniqueness of Tail Attacks from previous research is that Tail Attacks aim to create very short (hundreds of milliseconds) resource contention (e.g., CPU or disk I/O) with dependencies among distributed nodes, while giving an "Unsaturated illusion" for the state-of-the-art IDS/IPS tools leading to a higher level of stealthiness.

The most challenging task for launching an effective Tail Attack is to understand the triggering conditions of millibottlenecks inside the target web system, and quantify their long-term damages on the overall system performance. To thoroughly comprehend the attack scenario, we exploit the traditional queueing network theory to model the n-tier system, and analyze the impact of our attacks to the end-users and the systems through two new proposed metrics: *damage length* during which the new coming requests can be dropped with highly probability, and *millibottleneck length* during which the bottleneck resources sustain saturation. To fit the dynamics of background requests and system state, we develop a feedback control framework. Given the implementation based on the feedback control algorithm, we can effectively control the attacks, and find that our attacks can not only achieve high attack efficiency, but also escape the detection mechanisms based on human-behavior models, which further increases the stealthiness of the attack.

In brief, this work makes the following contributions:

- Proposing Tail Attacks by exploiting resource contention with dependencies among distributed nodes, that can significantly cause the long-tail latency problem in web applications while the servers are far from saturation.

- Modeling the impact of our attacks on n-tier systems based-on queueing network theory, which can effectively guide our attacks in an even stealthy way.
- Adopting a feedback control-theoretic (e.g., Kalman filter) framework that allows our attacks to fit the dynamics of background requests and system state by dynamically tuning the optimal attack parameters.
- Validating the practicality of our attacks through not only analytical, numerical, and simulation results but also experimental results of a representative benchmark website equipped with state-of-the-art DDoS defense tools in real cloud production settings.
- Presenting a conceptual solution to detect and defense the proposed attacks, involving three stages: fine-grained monitoring, identifying bursts, and blocking bots.

We outline the rest of this chapter as follows. Section 3.2 describes an attack scenario and the practical impact of Tail Attacks in real cloud production settings. Section 3.3 models our attack scenarios in an n-tier system using queueing network theory, and provides an effective approach to solve the potential optimal attack parameters numerically. Further, we evaluate the attack analytical model in JMT [63] simulator environment and suggest several guidelines to choose the optimal attack parameters in more complex cases (e.g., the competition for free slots of a queue between attack requests and normal requests, overloaded attack requests can be also dropped by the front-tier server). Section 3.4 describes our concrete implementation to launch Tail Attacks in real web applications. We adopt Kalman filter, a feedback control-theoretic tool, to automatically adjust the optimal attack parameters fitting the dynamics of target system state(e.g., dataset size change) and background workload. Section 3.5 shows our attack results of RUBBoS [30] benchmark website we have conducted in real cloud production settings, which further confirm the effectiveness and stealthiness of the proposed attacks, and the practicality of the control framework in more practical Web environments. Section 3.6 provides a "tit-for-tat" strategy to detect and defend our attacks targeting the unique scenario and feature of the proposed attack.

Section 3.7 discusses some additional factors that may impact the effectiveness of Tail attacks. Section 3.8 presents the related work and Section 3.9 concludes the chapter.

3.2 Scenario and Motivations

Consider a scenario of a Tail attack in an n-tier system in Figure 3.1. By alternating short "ON" and long "OFF" attack burst, an attacker guarantees the attack both harmful and stealthy. Short "ON" attack burst is typically on the order of milliseconds. The following sequence of causal events will lead to the long-tail problem at moderate average utilization levels during the course of Tail Attacks. (Event1) The attackers send intermittent bursts of attack but legitimate HTTP requests to the target system during the "ON" burst period; each burst of attack requests are sent out within a very short time period (e.g., 50ms). (Event2) Resource millibottlenecks occur in some node, for example, CPU or I/O saturates for a fraction of a second due to the burst of attack requests. (Event3) A millibottleneck stops the saturated tier processing for a short time (order of milliseconds), leading to fill up the message queues and thread pools of the bottleneck tier, and quickly propagating the queue overflow to all the upstream tiers of the n-tier system as shown in Figure 3.1,b. (Event4) The further incoming packets of new requests are dropped by the front tier server once all the threads are busy and TCP buffer overflows in the front tier. (Event5) On the end-user side, the dropped packets are retransmitted several seconds later due to TCP congestion control (minimum TCP retransmission time-out is 1 second [39]), the end users with the requests encountering TCP retransmissions perceive very long response time (order of seconds). Long "OFF" attack burst is typically on the order of seconds, in which the target system can cool down, clearing up the queued requests and returning back to a low occupied state shown in Figure 3.1,a. Unlike the traditional flooding DDoS attacks which aim to bring down the system, our attack aims to degrade the quality of service by causing the long-tail latency problem for some legitimate users while keeping the attack highly stealthy. The alternating short "ON" and long "OFF" attack burst can effectively balance the trade-off between attack damage and elusiveness.

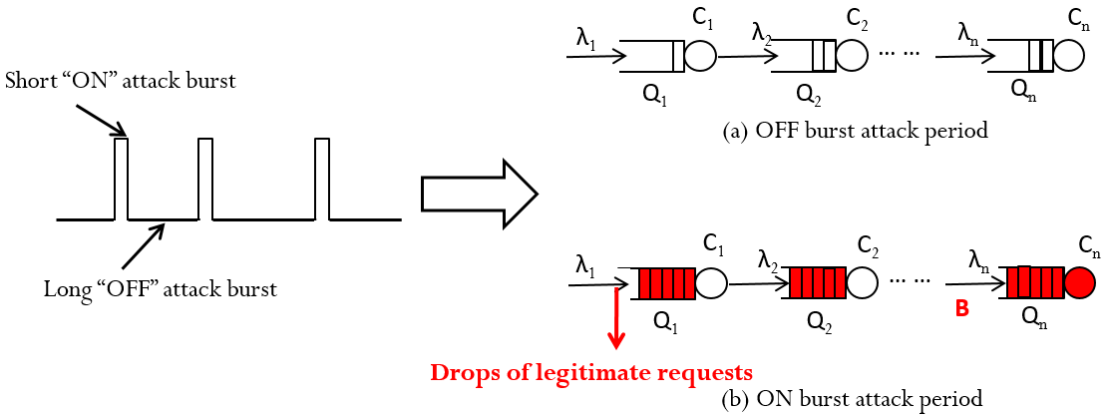


Figure 3.1. Attack scenario and system model of Tail attacks alternating short "ON" and long "OFF" attack burst.

Table 3.1 shows the impact of Tail Attacks through concrete benchmark web application with real production settings deployed in the most popular two commercial cloud platforms (Amazon EC2 [64], Microsoft Azure [65]) and one academic cloud platform (NSF Cloudlab [45]). We use a notation CloudPlatform-ServerTiers-BaselineWorkload to denote the cloud platform, the configuration of the n-tier system, and the background workload. For Server Tiers, We use a four-digit (or three-digit) notation #W#A#L#D to denote the number of web servers, application servers, load-balance servers (may not be configured), and database servers. More experimental details are available in Section 3.5.1. Table 3.1 compares the tail latency of the target system under attack and without attack, indicating the significant long tail latency problem under attack. Such long tail latency problem (e.g., 95th percentile response time $>$ 1 second) is considered as severe performance degradation by most modern e-commerce web applications (e.g., Amazon) [18–21,58]. At the same time, the average response time is still in acceptable range under attack, making the illusion of "business as usual" for system administrators.

3.3 Tail Attacks Modeling

In this section, we provide a simple model to analyze the impact of our attacks to the end-users and the victim n-tier system. Based-on the simplified model we introduce an effective approach of getting the potential optimized attack parameters to achieve our

Table 3.1. The measured long-tail latency under Tail Attacks in real cloud production setting. W/o att.: Without attacks, Att.: under Tail Attacks, RT: response time(ms).

Setting	95ile RT		98ile RT		99ile RT		Average RT	
	W/o att.	Att.	W/o att.	Att.	W/o att.	Att.	W/o att.	Att.
EC2-111-2K	255	1347	267	1538	308	1732	192	273
EC2-1212-4K	247	1139	282	1341	328	1683	170	218
EC2-1414-6K	263	1085	270	1285	312	1628	160	206
Azure-111-1K	251	1153	274	1295	295	1821	176	290
Azure-1212-2K	254	1090	278	1284	295	1507	177	221
Azure-1414-3K	264	1090	297	1314	325	1510	181	242
NSF-111-1K	141	1217	151	2262	159	7473	90	327
NSF-1212-2K	128	1101	143	1502	167	7016	88	309
NSF-1414-3K	118	1014	122	1413	127	3152	71	268

attack goal. Finally, we evaluate the model via simulation experiments and suggest several approaches to tune the attack parameters.

3.3.1 Model

Queueing network models are commonly used to analyze the performance problems in complex computer systems [66], especially for performance sizing and capacity provision. Here, we use a well-tuned queuing network to model n-tier systems, and analyze the sequence of causal events and the impact in the context of Tail Attacks shown in Figure 3.1. Table 3.2 summarizes the notation and description of the parameters used in our model.

The basic attack pattern (see Event1 in Section 3.2) shown in Figure 3.1 is that during the "ON" burst period (L) the attackers send a burst of attack requests with the rate (B), after the "OFF" burst period (T-L) they send another burst again, and repeat this process during the course of a Tail attack. If all the attack requests will not be dropped by the target system (more complex case will be discussed in Section 3.3.3), we can calculate the attack volume during a burst by:

$$V = B * L \tag{3.1}$$

We assume that the external burst of legitimate HTTP requests (Event1 in Section 3.2) can cause sudden jump of resource demand flowing into the target system and cause milli-

Table 3.2. Notation and description of the parameters used for modeling and analyzing Tail Attacks.

Parameter	Description
Q_i	the queue size for the ith tier
$C_{i,A}$	the capacity serving attack requests for the ith tier
$C_{i,L}$	the capacity serving legitimate requests for the ith tier
λ_i	the legitimate request rate terminating in the ith tier
B	the attack request rate during a burst
L	the burst length during a burst
V	the burst volume during a burst
T	the interval between every two consecutive bursts
l_i	the time to fill up the queue of the ith tier
P_D	the period of the requests dropped during a burst
P_{MB}	the period of a millibottleneck during a burst
$\rho(L)$	the average drop ratio during a burst
$\rho(T)$	the drop ratio during the course of an attack

bottlenecks (Event2 in Section 3.2) in the weakest point of the system [33]. In our model analysis, we assume that the n-th tier is the bottleneck tier. For example, the bottleneck typically occurs in the database tier (the n-th tier) in web applications due to the high resource consumption of database operations.

Due to the inter-tier dependency (call/response RPC style communication) in the n-tier system, one queued request in a downstream server holds a thread in every upstream server. Thus, the system administrator typically configures the queue size of upstream tiers bigger than the queue size of downstream tiers. In this case, millibottlenecks (Event2 in Section 3.2) caused by overloaded attack bursts can lead to cross tier queue overflow from downstream tiers to upstream tiers (Event3 in Section 3.2) due to the strong dependency among n-tier nodes. If the queue size satisfies

(C1) $Q_1 > Q_2 > ... > Q_{n-1} > Q_n$

and the burst rate satisfies

(C2) $\lambda_n + B > C_n$

for all i=1,...,n, then the time needed to fill up the queue for the n-th server is approximately

$$l_n = \frac{Q_n}{(\lambda_n + B - C_{n,A})} \tag{3.2}$$

$$l_{n-1} = \frac{(Q_{n-1} - Q_n)}{(\lambda_{n-1} + \lambda_n + B - C_{n,A})} \tag{3.3}$$

...

$$l_1 = \frac{(Q_1 - Q_2)}{(\sum_{i=1}^{n} \lambda_{\ i} + B - C_{n,A})} \tag{3.4}$$

When millibottlenecks occur in the n-th server, firstly the queue in the n-th tier is overflown during l_n, which equals the available queue size of the n-th tier divided by the newly-occupied rate for the queue of the n-th tier in Equation 3.2. The available queue size equals the queue size of each tier subtracting the queue size of its directly downstream tier. The newly-occupied rate equals the incoming rate of each tier subtracting the outgoing rate of the total system ($C_{n,A}$), the incoming rate of the n-th tier includes the requests going through the n-th tier and terminating in the n-th tier (B for the attack requests and λ_n for the normal requests). We carefully choose the attack requests [59] guaranteeing the attack requests go through every tier and terminate in the last bottleneck tier (detailed implementation in Section 3.4.2). Equation 3.3 represents the time to fill up the queue in the *n-1*-th tier. Because one queued request in a downstream server holds a position in the queue of every upstream server, after the n-th tier is full, the available queue size of the *n-1*-th tier should be (Q_{n-1} - Q_n). All the requests arriving to a downstream tier need to go through every upstream tier, thus the incoming rate of the *n-1*-th tier includes the request rate of terminating in the *n-1*-th tier (λ_{n-1}) and the request rate of going through the *n-1*-th tier (λ_n + B). Similarly, we can calculate the time to fill up every queue in the n-tier system during the process of propagating the queue overflown. Finally, the required time to overflow all the queues in the n-tier system is the sum of l_i.

Once all the queues are overflown (Event3 in Section 3.2) in the n-tier system, the new incoming requests may be dropped by the front tier (Event4 in Section 3.2). We term the period of the requests dropped during a burst as *damage length*. If the attackers continue to send attack requests to the system with overflown queues, and we assume that attack requests can always occupy the free position of the queue in the system (more complicated case will be discussed in Section 3.3.3), then we can approximately infer *damage length* by:

$$P_D = L - \sum_{i=1}^{n} l_i \tag{3.5}$$

Further, the end-users with the dropped requests perceive very long response time (Event5 in Section 3.2), leading to the long-tail latency problem caused by our attacks (Event1 in Section 3.2). Because long response time resquests occur during damage length, percentile response time can be approximately estimated as follows,

$$\rho(\mathrm{T}) = \frac{P_D}{T} \tag{3.6}$$

During a burst, the servers need to provide all the required computing resources (including bottleneck resources) to serve both attack requests and normal requests. We term the period of a millibottleneck during a burst as *millibottleneck length*, during which bottleneck resources sustain saturation (e.g., transient CPU satuation). Thus, *millibottleneck length* should involve the resource consumption to serve both attack and normal requests during a burst.

During *millibottleneck length*, the bottleneck resources sustain saturation. *Millibottleneck length* should involve the serving time for both attack and normal requests during a burst. The attackers only send requests within the short "ON" period, thus the amount of attack requests is the burst volume V. Meanwhile, the legitimate users always send requests during *millibottleneck length*, thus the saturation length is an infinite recursive process until it converges to zero exponentially. Equation (3.7) represents *millibottleneck length* derived

through the geometric progression in mathematics, here, m limits to infinity, P_{MB} is a function of V.

$$\begin{aligned} P_{MB} &= V * \frac{1}{C_{n,A}} + V * \frac{1}{C_{n,A}} * \lambda_n * \frac{1}{C_{n,L}} \\ &+V * \frac{1}{C_{n,A}} * \lambda_n * \frac{1}{C_{n,L}} * \lambda_n * \frac{1}{C_{n,L}} + ... \\ &+V * \frac{1}{C_{n,A}} * (\lambda_n * \frac{1}{C_{n,L}})^m \\ &= \lim_{m\to\infty} \sum_{k=0}^{m} V * \frac{1}{C_{n,A}} * (\lambda_n * \frac{1}{C_{n,L}})^k \\ &= V * \frac{1}{C_{n,A}} * \lim_{m\to\infty} \frac{(1-(\lambda_n * \frac{1}{C_{n,L}})^{m+1})}{(1-(\lambda_n * \frac{1}{C_{n,L}}))} \\ &= V * \frac{1}{C_{n,A}} * \frac{1}{(1-(\lambda_n * \frac{1}{C_{n,L}}))} \end{aligned} \tag{3.7}$$

where $1/C_{n,A}$ and $1/C_{n,L}$ are the service time for attack and normal requests in the bottleneck tier, respectively.

3.3.2 Numerically Solve Attack Parameters

Based on the model, we can infer the damage and elusiveness of our attacks through damage length and millibottleneck length. Further, if we assign the attack goal and know system parameters, we can calculate the optimal attack parameters mathematically.

To get some reasonable constant parameters $(\lambda, C_{i,A}, C_{i,N}, Q)$ in the model, we estimate these constants via profiling the service time of each type of request of each component tier in the benchmark web-site RUBBoS [30](more details in Section 3.5.1), the capacity of each tier C_i can be calculated from the service time. We choose *heavy requests* (e.g., long service time by consuming more system bottleneck resource, detailed explanation in Section 3.4.2) as attack requests. Table 3.3 lists a group of reasonable values of the constants for our model profiled in RUBBoS. During the profiling, we choose 2000 legitimate users with 7-second think time as our baseline experiment. All the transactions supported by RUBBoS are terminated in MySQL, each transaction follows a static page-load terminated

Table 3.3. Constant parameters estimation profiled in RUBBoS experiment with 2000 concurrent users.

Server	Tier (i)	λ_i	$C_{i,A}$	$C_{i,L}$	Q_i
Apache	1	280	3443	3657	55
Tomcat	2	0	1300	1987	25
MySQL	3	280	280	725	6

in Apache, and no transaction terminates in Tomcat. Thus, the request rate of each tier λ_i is 280, 0, and 280, respectively. We set the queue size of each server Q_i satisfying the condition *C1* in Equation (3.2).

Suppose that we set our attack goal as 95th percentile response time longer than 1 second which is a severe long-tail latency problem for most e-commerce websites [18–21], and the duration of a millibottleneck less than 0.5 seconds in the bottleneck tier such that the average utilization can be at moderate level (e.g., 50-60%) to bypass the defense mechanisms. If we assume the burst interval T is 2 seconds, then the input to the model can be two inequations: damage length P_D is bigger than 0.1 seconds and millibottleneck length P_{MB} is less than 0.5 seconds. Further, we turn these two inequations to L as a function of B, the others parameters are all constants(Inequation (3.8) and (3.9)).

$$\begin{aligned} L >= \sum_{i=1}^{n} l_i + 0.1 = & \frac{Q_n}{(\lambda_n + B - C_{n,A})} \\ & + \frac{(Q_{n-1} - Q_n)}{(\lambda_{n-1} + \lambda_n + B - C_{n,A})} \\ & + ... + \frac{(Q_1 - Q_2)}{(\sum_{i=1}^{n} \lambda_{\ i} + B - C_{n,A})} + 0.1 \end{aligned} \tag{3.8}$$

$$L <= 0.5 * (1 - \lambda_n * \frac{1}{C_{n,L}}) * \frac{C_{n,A}}{B} \tag{3.9}$$

Thus, the problem of selecting a set of optimal attack parameters (B,L, V, T) can become a nonlinear optimization problem. Although nonlinear optimization problem is hard to solve since there exist multiple feasible regions and multiple locally optimal points in those

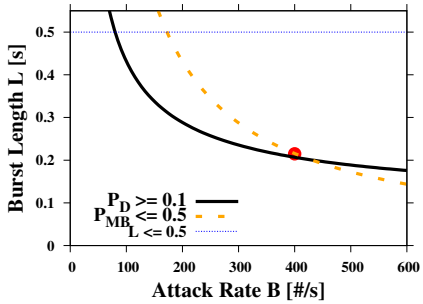
regions, we can add more constraints to narrow the range of feasible regions. For instance, the burst length obviously should be less than target millibottleneck length (e.g., burst length L less than 0.5 seconds).

Substituting the constant parameters in Inequation (3.8) and (3.9), we can get an unique feasible region as the potential attack parameters shown in Figure 3.2, a. In real cloud production settings, the queue size must be diverse according to the capacity of the websites. In Figure 3.2, b, we can see that as the queue of the front tier (e.g., Apache) increases from 30 to 80, the feasible region reduces; when it increases to 130 (the red line), the two inequations do not overlap, which implies that there is no solution to satisfy our predefined attack goal. The fundamental reason is that our attack goal is too strict, which seems to be an impossible mission. Note that when the queue of the front tier is 80, in the strictest attack target cases ($P_D >= 0.2$ seconds, the red line in Figure 3.2, c; or $P_{MB} <=$ 0.3 seconds, the red line in Figure 3.2, d), there is also no solution that can solve the attack parameters of our attacks. We will further discuss how to deal with the non-solution cases in Section 3.4.1.

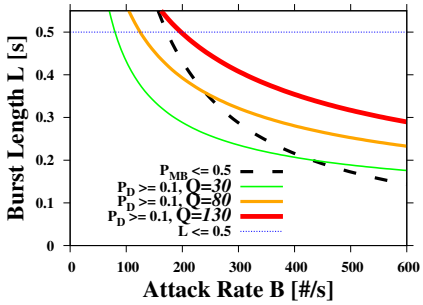
3.3.3 Simulation Experiments

The numerical solver in the previous section does not consider many aspects of the real system (e.g., the competition of the free position in the queue between attack requests and normal requests, overloaded attack requests can be dropped, etc.). To further validate the simple model, we present results from Java Model Tools(JMT) [63] in which such limitations are absent. JMT is an open source suite for modeling Queuing Network computer systems. It is widely used in the research area of performance evaluation, capacity planning in n-tier systems. Thus, it is a natural choice to evaluate the impact of our attacks in n-tier systems. We modify the JMT code and simulate the bursts of attack requests for our attacks with the configurable attack parameters in our model.

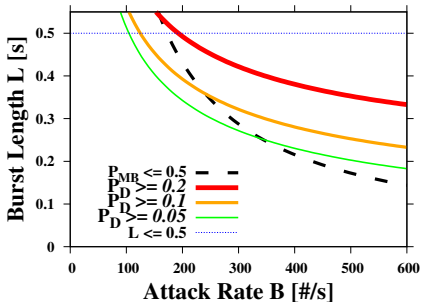
Given the proposed model and the idea of solving the nonlinear optimization problem, we can get the feasible region of attack parameters. We initialize the parameters in



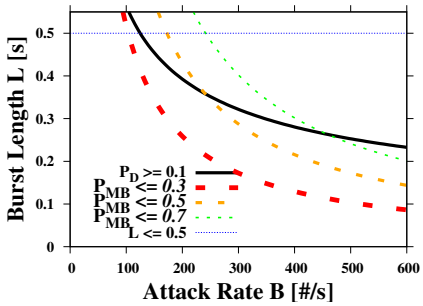
(a) There exists an overlapped feasible zone below the dash line Millibottleneck length and above the solid line Damage length.



(b) As Q increases, the feasible zone narrows down until no solution when Q is 130 (red line).



(c) Various target Damage length. No solution when Damage length is $>$ 0.2s (red line), since no overlap exists.



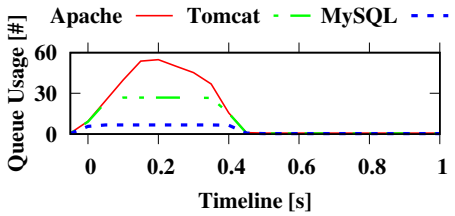
(d) Various target Millibottleneck length. No solution when Millibottleneck length is $<$ 0.3s (red line), since no overlap exists.

Figure 3.2. Numerically solve the optimal attack parameters. Solid line depicts damage length, the target zone is above the solid line; and dash line depicts millibottleneck length, the target zone is below the dash line.

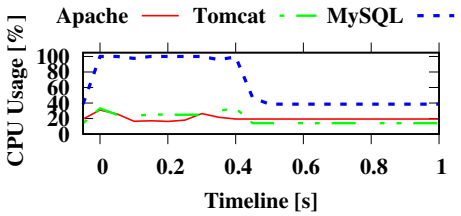
JMT similar to the setting of our numerical solver, and choose a potential optimal point (400,0.215) in Figure 3.2, a as our attack parameters, the attack rate B is 400 requests per second and the burst length L is 0.215 seconds, so the attack volume per burst V is 86 (see equation (3.1)) if all the attack requests will not be dropped by the target system.

Figure 3.3 shows the results of one burst during 1 second time period using fine-grained monitoring (e.g., 50 milliseconds) in JMT expriment. Figure 3.3, a illustrates the process of filling up all the queues in the n-tier system. Note that the queue of MySQL, Tomcat, and Apache is overflow from down-stream tiers to up-stream tiers overtime. The CPU saturations of the bottleneck tier MySQL last approximately 400 milliseconds as shown in

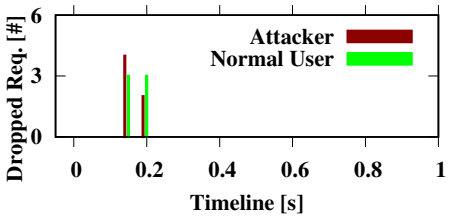
Figure 3.3, b, less than the expected value 500 milliseconds calculated by our model, since overloaded attack requests are also dropped by the front tier which will not go through the front tier and into the bottleneck tier. Figure 3.3, c shows dropped requests perceived by attackers and legitimate users in the corresponding burst. Note that the dropped requests span two sampling duration (100 milliseconds), validating our model expectation for the dropped length. The interesting observation is that the dropped requests from attackers is bigger than ones from legitimate users at the time of 0.15, the opposite phenomenon happens in the next sampling windows at the time of 0.2. This implies that the requests from the attacker and ones from the legitimate users compete the available position of the queue freed by the outgoing request in the n-tier system during *damage period* [62], eventually the loser will be dropped.



(a) Process of filling up the queues in 3-tier system, queue overflow are propagated from the bottleneck tier (MySQL) to the upstream tiers (Tomcat, then Apache).



(b) Millibottleneck length (MySQL CPU) is less than the expected value 500ms, since overloaded attack requests are also dropped by the front tier (Apache).



(c) The shift of the amount of dropped requests during damage length shows the competition for the available slot of the queue freed by the outgoing requests.

Figure 3.3. Results of one burst during 1 second period using fine-grained monitoring (50 milliseconds) in JMT experiment.

However, when we aggregate the data of the legitimate users during the 3-minute simulation experiment, the amount of dropped requests is 1099 and the total requests is 54901, thus the actual drop ratio for the legitimate users is 2%, which is far from the predefined goal of the drop ratio 5%. From this observation, we should calibrate the drop ratio from Equation 3.6 to

$$\rho(\mathrm{T}) = \frac{P_D * \rho(\mathrm{L})}{T} \tag{3.10}$$

Here, ρ(L) refers to the average drop ratio for the requests of the legitimate users during *damage length*, which represents the competition ability for attack requests compared to normal requests. In the previous JMT experiment, ρ(L) is approximately 0.4.

Due to the existence of the competition between the attackers and the legitimate users, the attackers may fail to get the predefined attack goal (e.g., 95th percentile response time $>$ 1 second) by using the recommended attack parameters of the proposed model. We further investigate how to increase the competition ability of attack requests, and the drop ratio of the requests from the legitimates users during damage length, namely ρ(L). Finally, the attackers can choose the optimal attack parameters to achieve high damage with low cost and high stealthiness. For simplicity, we assign attack interval T as fixed value (say, 2 seconds), since our focus on this chapter is to investigate how to effectively trigger the millibottlenecks which is predominantly determined by the other three parameters (B,L,V). Due to interdependent relationship of these three parameters in Equation 3.1, we fix one parameter (L or V), then observe the impact with various attack rate B. We still consider the marked potential optimal point (400,0.215) in Figure 3.2, a as our baseline attack parameters.

First, we fix burst volume V as 86, and select a set of attack rate (from 300 to 800) to conduct our attack experiments using JMT. Table 3.4 shows that as the attack rate increases, the drop rate *$\rho(T)$* accordingly increases, which confirms that higher attack request rate, compared to normal request rate, can achieve higher competition ability to seize the available position in the queue.

Table 3.4. Fix burst volume V=86. Bigger B higher $\rho(T)$, implying higher competition ability for the burst with larger B.

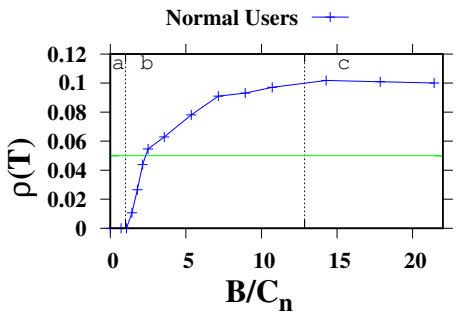
Attack Para.		Legitimate Users		
B	L	Dropped Reqs.	Total Reqs.	ρ(T)
300	0.285	1096	55998	0.0196
400	0.215	1099	54901	0.0200
500	0.173	1705	54292	0.0314
600	0.144	2082	53915	0.0386
700	0.123	2326	53671	0.0433
800	0.108	2399	53598	0.0448

Next, we fix burst length L as 0.215 seconds, and select another set of attack rate to conduct our JMT experiments. Figure 3.4 depicts $\rho(T)$ and $\rho(L)$ as a function of the ratio of attack rate and service rate for the bottleneck tier. We mark two vertical lines to split up into three zones with various attack rate B. (1) In zone a, B is less than $C_{n,A}$, the drop ratios are all zero, since the attack rate is too low to trigger an effective millibottleneck to lead to cross tier queue overflown in the target system, which violates the condition *C2* in Equation 3.2. (2) In zone b, B is bigger than $C_{n,A}$, the drop ratio increases non-linearly as B increases. Observe that ρ(L) of Normal Users is a little bit bigger than ρ(L) of Attackers, because B is bigger than λ_n. In this case, the requests from the attackers can seize the available position in the queue with a more highly probability than ones from normal users. (3) In zone c, B is bigger than $C_{1,A}$, the attack requests are directly dropped by the most front tier, thus the increase of B does not contribute to the drop ratio of the legitimate users, it only increases the drop ratio itself. Given this observation, we should choose a moderate B until the attack goal is achieved (e.g., the green horizontal line in Figure 3.4).

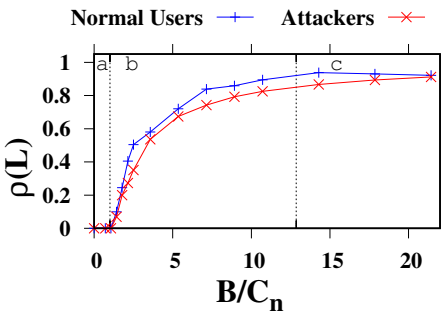
3.4 Tail Attacks Implementation

3.4.1 Overview

As mentioned before, the analytical model used in the numerical solver analyzes the simple scenario skipping many aspects of the system reality (e.g., the competition for the free position of the queue between attack requests and normal requests, overloaded attack



(a) $\rho(T)$ as a function of the ratio of attack rate and service rate



(b) $\rho(L)$ as a function of the ratio of attack rate and service rate

Figure 3.4. Fix burst length L=0.215 seconds. (1) in Zone a, $B < C$n, B is too low to trigger effective millibottlenecks; (2) in Zone b, $B > C$n , the bottleneck is in n-th tier, $\rho(T)$ increases as B increases; (3) in Zone c, $B > C1$, the attack requests are directly dropped by the most front tier, no obvious attack effect ($\rho(T)$ is flat) even though B increases tremendously, implying that we should choose a moderate B until the attack goal is achieved (e.g., the green horizontal line is a target).

requests can be dropped, etc.), and the simulation experiments show more complicated cases involving the absent system reality. However, our attacks do not consider more realistic case with dynamics of baseline workload (e.g., for e-commerce applications, the baseline workload during the day time is usually significantly higher than that during the mid-night period) or system state(e.g., dataset size change). For example, the peak workload occurs at approximately 1:00 p.m. during the week of Thanksgiving [1]. A set of effective attack parameters of Tail Attacks may become failed ones over time, either it can not trigger millibottlenecks (not enough attack requests) or it might trigger the defense alarm of the target system (too frequent or massive attack requests). How can we dynamically adjust the attack parameters catering to instant system state and baseline workload is a big challenge for Tail Attacks. The static attack parameters in the dynamical environment may make the attack either invalid like a "mosquito bite" or easily exposed to the detection mechanisms. In this section, we implement a feedback control-theoretic (e.g., Kalman filter) framework that allows attackers to fit the dynamics of background requests or system state by dynamically adjusting the optimal attack parameters in Figure 3.5.

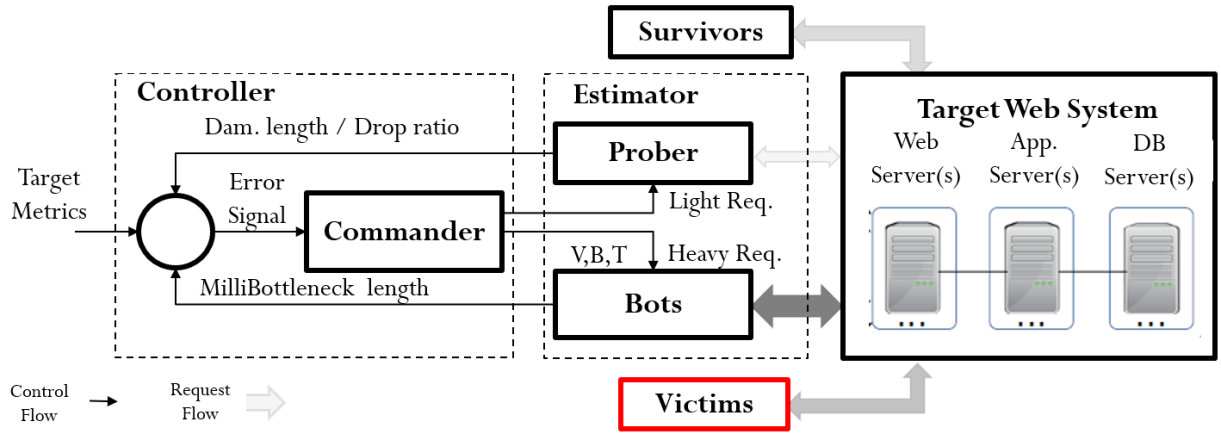


Figure 3.5. A feedback control framework which can launch effective Tail attacks to fit the dynamics of background requests or system state by dynamically adjusting the optimal attack parameters.

Via our best practice, we find that the attack rate should not be invariable to maximize the attack effectiveness and stealthiness. Kuzmanovic et al. [37] suggest a double-rate DoS steam to minimize the attack cost. We design a three-stage transmitting strategy to send one burst: Quickly-Start, Steadily-Hold and Covertly-Retreat. In Quickly-Start stage, the attacker sends the burst of requests at a high rate to quickly fill up all the queues in the n-tier system, *heavy requests* (detailed explanation in Section 3.4.2) are preferred because it can consume more bottleneck resources and occupy the queue longer with low cost and high stealthiness, the amount of heavy requests during this stage should be large enough to temporarily saturate the bottleneck resource in the target system [59]. In Steadily-Hold Stage, the attackers should try their best to guarantee that the queue can be overflown during this stage and attack requests can seize the free position in the queue with highly probability. *Heavy requests* are not necessary to serve as attack requests during this stage. We prefer *light requests* to hold on queue, such that in the last Covertly-Retreat stage, the attack requests can quickly and covertly leave the systems. In Covertly-Retreat stage, there is no attack request to be sent out. Figure 3.6 demonstrates the queue shifts during a three-stage burst. Through the strategy of variable attack rate and various attack requests, we can solve the insolvable cases in Section 3.3.2 by carefully choosing less heavy requests as attack requests.

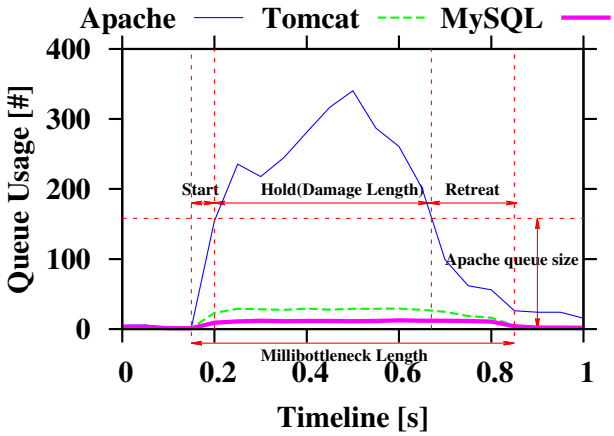


Figure 3.6. Queue shifts during a three-stage burst (quickly-start, steadily-hold, covertly-retreat).

Return back to our model in Equation 3.5, we can see the relationship between P_D and the attack rate B were nonlinear. We can transfer P_D as a function of V and B using equation 3.1:

$$P_D = \frac{V}{B} - \sum_{i=1}^{n} l_i \tag{3.11}$$

If we fix the attack rate B, mathematically, P_D and the attack volume V have a linear relationship; the same as P_{MB} (see Equation 3.7). These linear relationships provide us with a firm theoretical foundation to dynamically adapt the optimal attack parameters fitting the changes of system state and baseline workload. The overall control algorithm can be described in Algorithm 1.

3.4.2 Estimator

The Estimator, as illustrated in Figure 3.5, estimates three critical metrics in the control algorithm implementing the proposed model: service time of the requests, damage length P_D, and millibottleneck length P_{MB}. We use a prober to monitor attacks and infer damage length P_D, coordinate and synchronize bots to launch attacks and infer millibottleneck length P_{MB}.

Service time of a HTTP request is the time that the target web system needs to process the request without any queuing delay. It is easy to calculate the end-to-end response time

Algorithm 1 Pseudo-code for the control algorithm

```
1: procedure AdaptAttackParameters
2:     AttackReqST ← EstimateServiceTime
3:     DamLen ← EstimateDamageLenByProber
4:     MBLen ← EstimateMilliBottleneckLenByBots
5:     if DamLen = 0 then
6:         /* can not fill up queue, increase B */
7:         B ← B + stepB
8:     else
9:         gapDamLen ← Abs(DamLen - targetDamLen)
10:        stepV ← gapDamLen/AttackReqST
11:        if DamLen > targetDamLen then
12:            /* reduce damage length by decreasing V */
13:            V ← V − stepV
14:        else if DamLen < targetDamLen then
15:            /* increase damage length by increasing V */
16:            V ← V + stepV
17:        else
18:            /* current values are the optimal parameters. */
19:        end if
20:    end if
21:    if MBLen > targetMBLen then
22:        /* set max V */
23:        Vmax ← targetMBLen/AttackReqST
24:        V ← Vmax
25:        /* choose less heavy requests as attack requests */
26:    end if
27: end procedure
```

of a request using two time stamp of sending requests and receiving responses [67], we term them *start-time* and *end-time* of a HTTP request. The end-to-end response time of a request equals the difference between *end-time* and *start-time*. Typically, the end-to-end response time of a HTTP request involves three parts: the network latency between the client and the target web application, the queuing delay in the n-tier system, and the service time of each server. The network latency can be measured using the *ping* command. When the target system is at low utilization, the queuing effect inside the target system can be ignored. Thus, we can approximately estimate the service time of any HTTP request supported by the target web system as the end-to-end response time subtracting

the network latency when the target system is in the time block with a low workload. Since the service time of the estimated request may drift over time (e.g., due to changes in the data selectivity and the network latency variation) in real applications, we measure the service time of a HTTP request multiple times and take the average.

Previous research results [9] show that the predominant part of the service time of a request is spent on the bottleneck resource in the system. We call the requests that heavily consume the bottleneck resource as *heavy requests* with long service time (e.g., the request querying multi-tables in the database) while those consume no or little bottleneck resource as *light requests* with short service time (e.g., static requests) [59]. Thus, the prober naturally exploits *light requests* to monitor the impact of the attacks since it can be more elusive under the radar without causing any alert of the target web system; and the bots can take *heavy requests* as candidate attack requests since it can be more efficient causing millibottlenecks and cross tier queue overflow. More advance technology about profiling *heavy requests* will be discussed in Section 3.7. Through profiling and exploiting heavy requests [59], Tail Attacks can transiently saturating the critical bottleneck resource (e.g., database CPU) of the target systems, which can saturate the critical resource of the system with much lower volume (thus less bots are needed) compared to that of traditional flooding DDoS attacks which usually try to fully saturate the target network bandwidth.

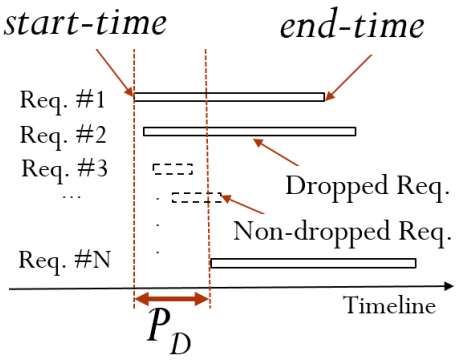
To estimate P_D, the prober needs to send probing requests (e.g., *light requests*) to the target system at a predefined rate and record *start-time* and *end-time* of a probing request. The recommended sending interval of probing requests is less than the target damage length, in the case, any probing request may not miss the period of overflown queue caused by an attack burst and the prober can sense P_D [35]. Figure 3.7, a illustrates the implementation approach to estimate P_D as *start-time* of the last dropped probing request subtracting *start-time* of the first dropped probing request during a burst. Since some probing requests may probably seize the free position in the queue and are not be dropped during the damage length, we can calibrate P_D by multiplying $\rho(\mathrm{L})$ (see equation 3.10).

Some websites may send some alarm to the users if they send the requests at a very high rate. In this case, the prober can send probing requests at an acceptable rate for the target web system and estimate drop ratio during a sampling period, then our control algorithm can exploit drop ratio as the target criterion to dynamically adjust the attack parameters.

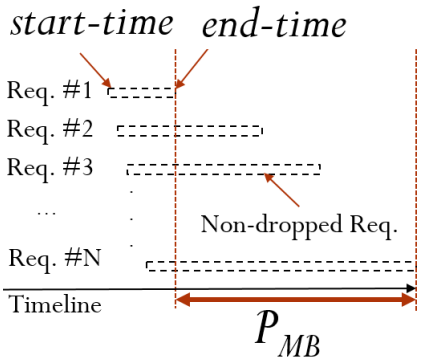
After sending a burst of attack requests (e.g., *heavy requests*) to the target web system, the bots can record *start-time* and *end-time* of an attack request and estimate P_{MB}. We only count the non-dropped attack requests, since the dropped request involves TCP retransmission time-out. There are two options to infer P_{MB}. One way is *end-time* of the last non-dropped attack request subtracting *start-time* of the first non-dropped attack request during one attack interval. As we mention before, the end-to-end response time of a HTTP request involves three parts: the network latency, the queuing delay, and the service time. In this way, the result overcharges a length of the network latency. The other option of inferring P_{MB} is *end-time* of the last non-dropped attack request subtracting *end-time* of the first non-dropped attack request during one attack interval shown in Figure 3.7, b. In this way, it undercharges a length of the service time. For the concrete environments of a special website, the attacker can choose any approach to estimate millibottleneck length. In our evaluation section, we choose the second one, since the network latency is much longer than the service time in our RUBBoS environment.

3.4.3 Controller

So far, we discuss many aspects that can influence the effectiveness of our attacks. In the model, the competition for the free slot of the queue between attack requests and normal requests may impact precision of damage length, and the dropped attack requests may decrease millibottleneck length. For Estimator, the network latency variation and the drifted target system state might reduce the accuracy of inferring damage length and millibottleneck length in our implementation. All of aspects lead to the observing and measuring inaccuracy, and result in the invalidation of launching an effective attacks using our control algorithm. To mitigate these negative impacts for our control algorithm, we



(a) Estimate damage length by start-time of the last dropped probing request subtracting start-time of the 1st dropped probing request during an attack burst.



(b) Infer millibottleneck length by end-time of the last non-dropped attack request subtracting end-time of the 1st non-dropped attack request during a interval.

Figure 3.7. Demonstration of inferring damage length and millibottleneck length by Estimator.

adapt a popular feedback-based control tool, Kalman filter [68]. On the one hand, it can take past measurements into account for implementing our feedback control algorithm, reducing the impact of the process noise (e.g., baseline workload and system state). On the other hand, it can mitigate the measurement noise due to inaccuracy of the estimator.

Let *z(k)* be the measurement of P_D in k-th burst by Estimator. Since P_D is a linear function of burst volume V (see equ. (3.11)), we can define *x(k)* using a linear dynamical system model:

$$SystemDynamics : x(k) = x(k-1) + U(k) + v(k) \tag{3.12}$$

$$MeasurementDomain : z(k) = x(k) + w(k) \tag{3.13}$$

where the variables *v(k)* and *w(k)* are the process noise (e.g., dynamics of baseline workload) and the measurement noise (e.g., imperfect estimation by Estimator), respectively. *U(k)* is the expected control result, a linear function of burst volume V.

Let $\hat{x}(k \mid k-1)$ be a priori estimate of state parameter x at burst k-th given the history of all *k-1* bursts, and let $\hat{x}(k \mid k)$ be a posteriori estimate of state parameter x at k-th burst. Further, let $P(k \mid k)$ be a posteriori error covariance matrix which quantifies the accuracy of the estimate $\hat{x}(k \mid k)$. The Kalman filter executes recursively for each new observation including two phases: Predict in which a priori estimate of state and error matrix are calculated, and Correct in which a posteriori estimate of state and error matrix are refined using the current measurement. The Kalman filter model for our control framework is given by:

Predict (Time Update)

$$\hat{\mathrm{x}}(k \mid k-1) = \hat{\mathrm{x}}(k-1 \mid k-1) + U(k) \tag{3.14}$$

$$\mathrm{P}(k \mid k-1) = \mathrm{P}(k-1 \mid k-1) + V(k) \tag{3.15}$$

Correct (Measurement Update)

$$\mathrm{K}g(\mathrm{k}) = \frac{\mathrm{P}(k \mid k-1)}{(\mathrm{P}(k \mid k-1) + W(k))} \tag{3.16}$$

$$\hat{\mathrm{x}}(k \mid k) = \hat{\mathrm{x}}(k \mid k-1) + \mathrm{K}g(\mathrm{k})(z(k) - \hat{\mathrm{x}}(k \mid k-1)) \tag{3.17}$$

$$\mathrm{P}(k \mid k) = (1 - \mathrm{K}g(\mathrm{k}))\mathrm{P}(k \mid k-1) \tag{3.18}$$

where *W(k)* and *V(k)* are the covariances of w(k) and v(k), respectively. In practice, we can estimate these two noise covariances using automatic mathematical tools (e.g., autocovariance least-squares method [69]) or manual observation to tune the optimal value. *Kg(k)* is termed the Kalman gain which represents the confidence index of the new measurement (*z(k)*) over the current estimate ($\hat{x}(k \mid k-1)$). If *Kg(k)* equals 1, it implies that the attacker

totally trust the measurement, the effectiveness of adjusting the attack parameters in our attacks is totally depending on the accuracy of the estimated value by Estimator.

Using the Kalman filter, the Controller in Figure 3.5 can predict the required attack parameters at *k-th* burst given the historical results of all *k-1* bursts, dynamically command the new parameters to the bots, and automatically and effectively launch Tail Attacks.

3.5 Real Cloud Production Evaluation

3.5.1 Tail Attacks in Real Production Settings

To evaluate the practicality of our feedback control attack framework in the real cloud production settings, we deploy a representative benchmark website in the most popular two commercial cloud platforms (Amazon EC2, Microsoft Azure) and one academic cloud platform (Cloudlab [45]).

We adopt RUBBoS [30], a representative n-tier web application benchmark modeled after the popular news website *Slashdot*. We configure RUBBoS using the typical 3-tier or 4-tier architecture. A sample setting *EC2-1414-6K* in Table 3.5 is 1 Apache web server, 4 Tomcat application servers, 1 C-JDBC clustering load-balance server, 4 MySQL database servers deployed in Amazon EC2, and 6000 concurrent legitimate users. RUBBoS has a workload generator to emulate the behavior of legitimate users to interact with the target benchmark website. Each user follows a Markov chain model to navigate among different webpages, with averagely 7-second think time between every two consecutive requests. Through modifying the RUBBoS source code, we simulate various baseline workload (e.g., variable concurrent users during the experiment). Meanwhile, in our experiments we adopt a centralized strategy to coordinate and synchronize bots [33,40,41] (more discussion about distrubuted bots coordination and synchronization in Section 3.7). All the bots are in the same location to rule out the impact of the shift of network-latency. We control a small bot farm of 10 machines (one of which serves as a centralized controller), synchronized by NTP services, which can achieve millisecond precision [70]. Each bot uses *Apache Bench* to send intermittent bursts of attack HTTP requests, commanded by our control framework.

All the VMs we run are 1 vCPU core and 2GB memory, which is the basic computing unit for the commercial cloud providers. We select HDD disk since our experimental workloads are browse-only CPU intensive transactions. We select t2.small instance ($0.023 per hour) in Amazon EC2 us-east-1a zone, and A1 ($0.024 per hour) instance in Microsoft Azure East US zone. They have similar prices and hardware configurations. However, the CPU core in EC2 (2.40GHz Intel Xeon E5-2676 v3) is more powerful than the one in Azure (2.10GHz AMD Opteron 4171 HE or 2.40GHz Intel Xeon E5-2673 v3). The worst one is in NSF Cloud (2.10GHz Intel Xeon E5-2450), where we run the VMs in Apt Cluster in the University of Utah's Downtown Data Center .

For the baseline workload, we have two chosen criteria: the bottleneck resource utilization (e.g., CPU utilization or Network bandwidth) is less than %50 (Column 6 and 8 of Table 3.5), and no long-tail latency problem exists in without-attacks cases shown in the Table 3.1. Because EC2 has more powerful CPU that we used in our experiments, it can serve higher baseline workload than the other two. The network overhead can be the bottleneck resource in EC2 platform even though CPU is at low utilizations [71]. In our experiments, MySQL CPU is the bottleneck tier in Azure and NSF Cloud due to the high resource consumption of database operations.

We pre-define our attack goal as the 95th percentile response time longer than 1 second and the utilization of the bottleneck resource less than 50%, and fix the attack interval as 2 seconds. Thus, we need to control damage length P_D longer than 100 milliseconds, millibottleneck length P_{MB} less than 500 milliseconds.

Column 2 to 5 of Table 3.5 show the corresponding model parameters in our real cloud production setting experiments controlled by our attack framework. It clearly shows that our attacks controlled by our algorithm can achieve the predefined targets (5% drop ratio ρ(T), damage length $P_D < 100$ms, millibottleneck length $P_{MB} < 500$ms). EC2 has more powerful CPU, so it requires more attack requests per burst V to launch a successful attack. As the servers scale out (from 111 to 1212, 1414), the n-tier system can service

Table 3.5. The corresponding parameters and bottleneck resource utilization of Tail Attacks in real production settings. W/o att.: Without attacks, Att.: under Tail Attacks, CPU: MySQL CPU usage, NW: Apache network traffic.

Setting	V (#)	P_D (ms)	ρ(T) (%)	P_{MB} (ms)	CPU (%)		NW (MB/s)	
					W/o att.	Att.	W/o att.	Att.
EC2-111-2K	202	96.4	5.64	287	18.8	27.3	116	167
EC2-1212-4K	217	96.9	5.47	297	16.0	20.2	239	282
EC2-1414-6K	234	105.3	5.89	271	15.2	18.1	352	401
Azure-111-1K	113	100.8	5.26	346	41.2	70.6	60	76
Azure-1212-2K	137	99.1	5.56	479	34.6	58.3	119	141
Azure-1414-3K	156	99.9	5.57	408	19.5	25.6	177	195
NSF-111-1K	97	101.0	5.40	453	49.1	76.8	58	72
NSF-1212-2K	112	96.7	5.75	490	48.4	62.4	118	137
NSF-1414-3K	131	99.4	5.22	470	34.6	51.0	173	186

more legitimate users, at the same time, in order to launch Tail Attacks it requires higher V due to their higher capacity, which means that bigger websites need larger botnet to attack. Column 7 and 9 of Table 3.5 show the average CPU utilization of MySQL tier and the average network traffic of Apache tier, which are the bottleneck resource in our experimental environment. Under our attacks in the real cloud production settings, the end users encounter the long-tail latency problem (see Table 3.1). However, all the bottleneck resources are under moderate average resource utilization even in large scale case (e.g., CPU is 25.6% in Azure-1414-3K, and NW is 401MB/s in EC2-1414-6K). These experimental results show that our attacks can cause significant long tail latency problem, while the servers are far from saturation, guaranteeing a high level stealthiness.

Figure 3.8 further illustrates the 3-minute detailed experimental results with various baseline workload under our attacks launched by our control framework in NSF-1212 setting. Figure 3.8, a shows baseline workload changes from 2000 to 500 concurrent users at the middle point of the experiment (each user has 7-second think time between two web-pages). Note that the shift of baseline workload in this case is different from the previous cases in which we scale out the servers. Figure 3.8, d shows the required burst volume

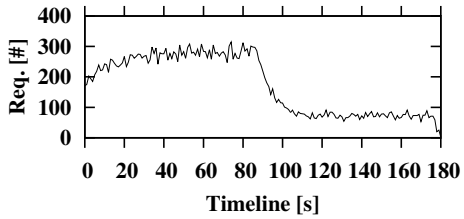
dynamically adjusted by our framework, a sudden increase at the middle due to the decrease of baseline workload. Figure 3.8, b and Figure 3.8, c depict damage length and millibottleneck length estimated by the prober and the bots, respectively. The measured average value of two critical metrics of our model are successfully controlled in the target range. Figure 3.8, e and Figure 3.8, f depict the drop ratio during every attack interval and the CPU utilization of MySQL using 1 second granularity monitoring, respectively, which match our predefined attack goal to a great extent. In general, through the RUBBoS experiments in real cloud production settings, we can validate and confirm the practicality of our attack control framework.

3.5.2 Tail Attacks under IDS/IPS systems

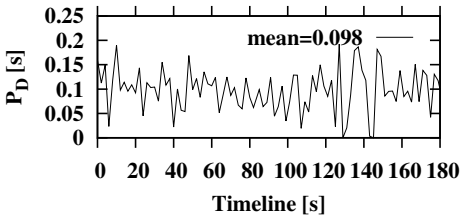
Next, in order to evaluate the stealthiness of Tail Attacks under the radar of state-of-the-art IDS/IPS systems, we deploy some popular defense tools in the web tier in our RUBBoS environments to evaluate whether our attacks can be detected by them.

Typically, the popular solution to mitigate the application layer DDoS attacks is identifying abnormal user-behavior [56, 57, 72–74]. Snort [43] is a signature rules based Open-Source IDS/IPS tool that is widely used in practice for DDoS defense, the users can customize the alert rules by setting reasonable thresholds in the specific systems. We set alert rules following some user-behavior models in Snort to evaluate whether our attacks deviate from the model (judging based on predefined thresholds). In the following experiments, we configure 2000 concurrent legitimate users, and our attack goal is to achieve the 95th percentile response time longer than 1s for these legitimate users.

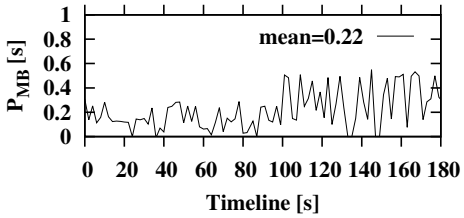
To validate the user behavior model, we take request dynamics model [44] as an example. The authors analyzed the distribution of a user's interaction with a Web server from a collection of Web server logs, categorized four session types (searching, browsing, relaxed and long session), and modeled the features for each type of session based on average pause between sessions. Typically, average inter-request interval for searching and browsing sessions is less than 10 seconds. RUBBoS [30] also models the inter-request inter-



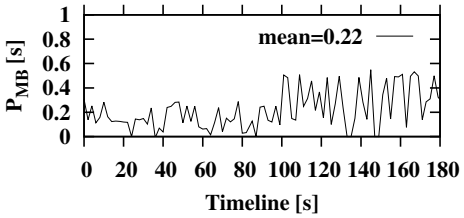
(a) The requests of baseline workload vary from 2000 to 500 concurrent users.



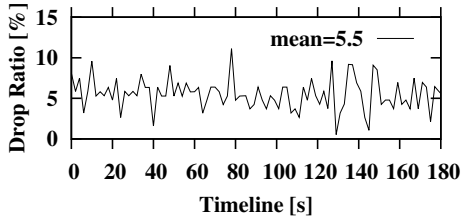
(b) Damage length inferred by the prober nearly equals the target 100ms.



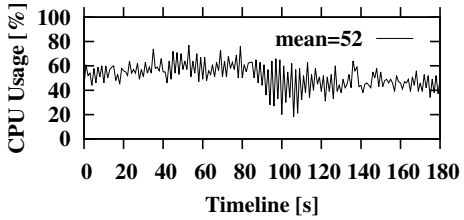
(c) Millibottleneck length estimated by the bots is less than 500ms, achieving the goal.



(d) Required attack volume V controlled by our framework increases as background requests decreases in Fig. 3.8, a.



(e) Drop ratio for legitimate users meets the target 5%, indicating the effectiveness controlled via damage length in Fig. 3.8, b.



(f) Bottleneck resource utilization approximately equals 50%, indicating the effectiveness controlled via millibottleneck length in Fig. 3.8, c.

Figure 3.8. Results of Tail Attacks on RUBBoS web application in a real cloud production setting

val per session as a Poisson process with the mean as 7, which means an average 7-second think time between two webpages for each session. In this case, we can calculate the 95% confidence interval as (2.814, 14.423). We can set the minimal boundary of the interval (e.g., rounded to 3, the statistical granularity is seconds in Snort) as the alert threshold to validate whether our attacks deviate the RUBBoS model, since the smaller alert threshold can lead to less false positive error. We use the "detection_filter" property in Snort to define the alert rules monitoring inter-request rate for each IP, which generate an alert once the amount of requests from the same IP exceeds the predefined threshold during a sampling statistical period. Column 1 and 2 of Table 3.6 depict the threshold and the sampling interval for these alert rules, respectively.

We use RUBBoS client to simulate 2000 concurrent legitimate users and craftily control our bots catering to the inter-request model in RUBBoS to bypass the above alert rules in Table 3.6 while achieving our attack goal. Due to the limited numbers of IPs, we can not have enough IP address to simulate the 2000 legitimate users and the bots from real IP address to evaluate the above detection rules in Snort. However, we can map a session to a individual IP address and implement the same detection algorithm using the "detection_filter" property of Snort to validate the above alert rules. We conduct a 3-minute successful attack experiment in our RUBBoS websites. In our case, to achieve the attack goal, the required attack parameters are V as 300, L as 50 and I as 3. To follow the inter-request model in RUBBoS (alert threshold as 3), it requires 301 totally-synchronized bots (one session simulates one bot in our experiments and sends one request in more than 3s interval) while avoid triggering the alerts (deviating from the model). Column 3 and 4 of Table 3.6 report the traced alert number from the bots and the legitimate users for these rules, respectively. As a result, our attacks can be totally invisible to these alert rules. Note that as the sampling interval increases the alerts from the legitimate users decrease, the reasonable sampling interval can reduce the false positive errors (typically the interval should be on the order of minutes). Another important guide to our attacks is that as the

sampling interval increases, the threshold of the rules also has to increase, which can give us more flexible options to send the attack requests using different intervals (e.g., in the above case, less than 10 requests every 30s interval per session, or less than 20 requests every 60s interval per session). To choose which sending pattern, we can further learn from the other user-behavior models to make our attacks even more stealthy.

Table 3.6. The attacker can successfully predict the inter-request model. Less the sampling period, higher alerts from the legitimate users, indicating higher false positive error. However, the attacker can trigger no alert by carefully controlling the request pattern.

Alert Rule Parameters		Triggered Alert Number	
Threshold	Sampling Period	Bots	Legitimate Users
1	3s	0	19262
5	15s	0	7032
10	30s	0	1074
15	45s	0	174
20	60s	0	23
50	150s	0	2
100	300s	0	0

Someone may argue that the "no alert" results are got from the assumption that the attackers comprehensively know the alert thresholds of the user-behavior model, such that they can design a corresponding attack pattern to avoid be detected. Most user-behavior models are public to both the defenders and the attackers, the only gap is the specific alert threshold and rules, which typically are learned from the server's past logs for their anomaly detection systems by the defenders of specific websites [44]. However, the attackers can use the questionnaire approach to similarly estimate the real users' behaviors, and use a conservative value as the potential threshold to design the attack pattern with the price of increasing more bots shown in Table 3.7. The four rows in Table 3.7 show four cases with different predicted values (3, 6, 12, 24). Obviously, as the predicted value is more conservative, it requires a bigger botnet (minimal size is 2400 when the prediction is 24). In the '24' case, the attacker must split the 2400 bots into 8 groups, each group takes turn to send a burst of 300 requests in every 24-second interval.

Table 3.7. The attacker conservatively predicts the inter-request model. To achieve a burst with 300 requests in every 3 seconds (V=300, L=50, I=3), more conservative predict needs bigger botnet. 1G.: 1Group, 2G.:2Groups, 4G.:4Groups, 8G.:8Groups, minS.:minimal Size, minG.:minimal Group.

Inter-request Model		Botnet		Designed Attack Pattern			
Actual	Predicted	minS	minG	1G.	2G.	4G.	8G.
3	3	300	1	1/3s	2/6s	4/12s	8/24s
3	6	600	2	-	1/6s	2/12s	4/24s
3	12	1200	4	-	-	1/12s	2/24s
3	24	2400	8	-	-	-	1/24s

3.6 Detection and Defense

Here, we consider a solution to detect and defend against Tail Attacks. There exists no easy approach to accurately distinguish the attack requests from legitimate requests. Instead, we can identify the attack requests by detecting the burst and matching the bursty arrivals to millibottlenecks and cross tier queue overflown (Event2 and Event3 in Section 3.2). We present a workflow to mitigate our attacks involving three stages: fine-grained monitoring, burst detection, and bots blocking.

The unique feature of our attacks is that we exploit the new-found vulnerability of millibottlenecks (with subsecond duration) in recent performance studies of web applications deployed in cloud [8–10]. In order to capture millibottlenecks, the monitoring granularity should be less than the millibottlenecks period in millisecond level. For example, if the monitoring granularity is 50ms, it can definitely pinpoint the millibottleneck longer than 100ms, probably can seize the millibottleneck in the range of 50ms to 100ms, but absolutely can not capture the millibottleneck less than 50ms. Thus, how to choose the monitoring granularity is depending on the observation and specific duration of the target bottlenecks.

Through fine-grained monitoring, we may observe a bunch of spike for each metrics (e.g., CPU utilization, request traffic, queue usage, etc.). However, our purpose is to detect the burst of attack requests, so we must discriminate the actual attack bursts from them. Based on the unique scenario of our attacks in Section 3.2, we can define our attack bursts

in which all the following events occur simultaneously: very long response time requests (dropped requests), cross tier queue overflown, millibottlenecks (e.g., CPU utilization, I/O waiting), and burst of requests. If all the events are observed in the same spike duration, we can regard the spike duration as a potential attack burst.

Once we identify the bursts of Tail Attacks, the next task is to distinguish the requests of the bots from the requests of the legitimate users during the burst and block them. The attacker, in our attack scenario, aims to coordinate and synchronize the bots to sending bursts of attack requests during short "ON" burst period and repeat the process after long "OFF" burst period as introduced in Section 3.2, we can ideally introduce a new request metric that quantifies the suspicion index of the incoming requests by aggregating the requests statistics during "ON" burst and "OFF" burst for further analysis. Specially, we define the *suspicion index* for each IP address as follows:

$$\mathrm{SI}_{IP} = \frac{\mathrm{NB}_{IP}}{\mathrm{N}_{IP}} \tag{3.19}$$

where NB_{IP} and N_{IP} are the number of requests for each IP during "ON" burst and the attack interval T (including "ON" and "OFF" burst), respectively. If SI_{IP} for a IP is close to 1, the IP is likely to be a bot; on the other hand, SI_{IP} of the legitimate user can be approximately the target attack drop ratio (e.g., 0.5 if the target is 95th percentile response time longer than 1 second). Figure 3.9 shows Probability Density Function of SI_{IP} in the RUBBoS experiment in Section 3.5.1. The red and green bars represent the suspicion index of the bots and the 2000 legitimate users, respectively. In this way to identify bots, the false positive and false negative error can be close to 0 with 100% high precision.

3.7 Discussions

3.7.1 Impact of load balancing

Some web applications adopt load balancing (e.g., Amazon Elastic Load Balancing [75]) in front of the system to distribute load across multiple web servers. This type of load

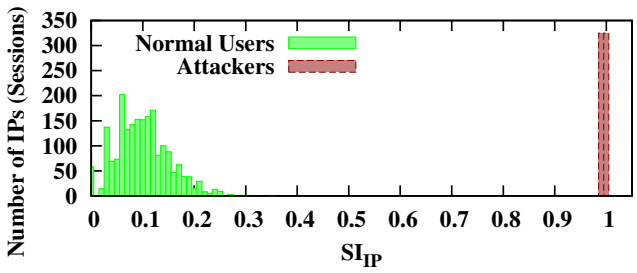


Figure 3.9. Distinguish the bots from the legitimate users via bi-modal suspicion index distribution.

balancing works well for stateless servers such as web servers, in which incoming traffic are supposed to evenly distribute among them. However, whether or not load balancing can mitigate the effectiveness of Tail attacks depends on the location of the bottleneck resource in the target web system. For example, if the bottleneck resource is in the web server tier, load balancing indeed increases the bar of an effective Tail attack since each web server only receives a portion of total attacking requests, thus higher volume of attacking requests per burst are needed to create millibottlenecks in the system. On the other hand, if the bottleneck resource is in a non-scalable tier such as the relational database tier, the load balancing in front tiers does not help mitigating the effectiveness of Tail Attacks. This is because no matter which web server an attacking HTTP request arrives at, the database queries sent out by the HTTP request may eventually converge to the same database server, and create millibottlenecks there.

3.7.2 Impact of cloud scaling

Large-scale web applications usually adopt dynamic scaling strategy (e.g., Amazon Auto Scaling [46]) for better load balancing and resource efficiency, however, Tail Attacks can easily bypass current dynamic scaling techniques, since the control window of the state-of-the-art scaling mechanism is usually in minute-level (e.g., Amazon CloudWatch monitoring in a minute granularity), while Tail Attacks are too short (sub-second duration) for them to catch and take any scaling actions [59]. The main advantage of Tail attacks is that it is invisible to most monitoring programs and can remain hidden for a long time

because of the low volume characteristic and the sub-second duration of millibottlenecks as shown in our experimental results.

3.7.3 Browser compatibility check

Some state-of-the-art defense tools may check the header information of each HTTP request to determine whether it is sent from a real browser or from a bot. A HTTP request sent from a real browser usually has completed header information such as "User-Agent", while such information may not appear or be difficult to be generated by a bot which only uses a script language to generate HTTP requests. In addition, some websites such as Facebook need a legitimate user to login first before any following transactions, especially *heavy query requests* (detailed explanations in Section 3.4.2). They may track the cookies stored in the client side browser in order to keep an active session in the server side; a bot may not be able to interact with such websites due to the lack of support of a real browser. We can address these challenges by using PhantomJS [76] to generate attacking HTTP requests. PhantomJS is a headless web browser without a heavy graphical user interface. It is built on top of WebKit, the engine behind Chrome and Safari. So PhantomJS can behave the same as a real browser does. Therefore, an attacker can launch browser-based Tail Attacks using heavy requests as attack requests by PhantomJS, and the generated requests will be extremely difficult to distinguish from the requests sent by legitimate users.

3.7.4 Distributed bots coordination and synchronization

One precondition of Tail attacks is that bots could be coordinated and synchronized so that the generated HTTP requests are able to reach the target web system within a short time window. Many previous research efforts already provide solutions, using either centralized [33, 40, 41] or decentralized methods [35], to coordinate bots to send synchronized traffic to cause network congestion at a specific network link. Centralized control can achieve higher level of bots coordination and synchronization, which enables a more effective Tail Attack compared to decentralized methods. In this chapter we adopt the

centralized control method to do experiments. On the other hand, a decentralized method in general is able to coordinate and synchronize more bots than a centralized one, thus making it possible to target large-scale/high-capacity websites. However, a decentralized method is more challenging to control the length of each burst of attacking requests arrived at the target website, thus mitigating the effectiveness of Tail Attacks.

3.8 Related Work

In this section, we review the most relevant work with regard to low-volume application-layer attacks, which is even stealthier to avoid traditional network-layer based detection mechanisms [6, 7, 77].

Low-volume DDoS application attacks are characterized by a small number of attack requests transmitted strategically to the target servers, as an extension of network-layer low-volume attacks [33–38, 60]. Macia et al. initially proposed *low-rate* attacks against application servers (*LoRDAS* [62]) that send traffic in periodic short-time pulses at a low rate, sharing the similar on-off attack pattern with our attacks. *Slow-rate* attacks [61] deplete system resources on the server's side by sending (e.g., slow send/Slowloris [78]) or receiving (e.g., slow read) traffic at a slow rate. Our attacks share the similar features of low attack volume with low-rate and slow-rate attacks. However, compared to these two attacks, our attacks dig more deeply into n-tier architecture applications while *LoRDAS* attacks only in 1-tier application server. Our analytical model for n-tier systems can guide and guarantee our attacks dynamically controlled in a more effective and elusive way by accurately estimating *damage length* and *millibottleneck length*. In addition, slow-rate attacks need to develop well-crafted HTTP header (Slow Headers) or body (Slow Body) thus expose obvious attack patterns to the defense tools, while Tail Attacks use the legitimate and normal heavy requests as our attack requests thus hide deeper. More importantly, we exploit the ubiquity of millibottlenecks (with sub-second duration) and strong dependencies among distributed nodes for web applications, leading to long-tail latency issue with a higher level of stealthiness than *LoRDAS* and *Slow-rate* attacks.

To mitigate application DDoS attacks, existing solutions typically focus on distinguishing application-layer DDoS traffic from the traffic created by legitimate users, such as abnormal user-behavior in high-level features [56, 72, 73] of surging web pages, in session-level [57], or in request-level [74]. To be stealthy, for the features of navigating web pages, our attacks can learn from the user behaviors of a legitimate user; as for session or requests level of our attacks, we can calculate the required optimized attack volume and botnet size discuss in Section 3.5. Compared to existing solutions, our proposed countermeasure can be tied to the unique feature of our attacks and accurately capture the bots.

3.9 Conclusion

We described Tail Attacks, a new type of low-volume application layer DDoS attack in which an attacker exploits a newly identified system vulnerability (millibottlenecks and resource contention with dependencies among distributed nodes) of n-tier web applications to cause the long-tail latency problem of the target web application with a higher level of stealthiness. To thoroughly comprehend the attack scenario (Section 3.2), we formulated the impact of our attacks in n-tier systems based-on queueing network model, which can effectively guide our attacks in a stealthy way (Section 3.3). We implemented a feedback control-theoretic (e.g., Kalman filter) framework that allows attackers to fit the dynamics of background requests or system state by dynamically adjusting the optimal attack parameters (Section 3.4). To validate the practicality of our attacks, we evaluated our attacks through not only analytical, numerical, and simulation results but also benchmark web applications equipped with state-of-the-art DDoS defense tools in real cloud production settings (Section 3.3 and Section 3.5). We further proposed a solution to detect and defense the proposed attacks, involving three stages: fine-grained monitoring, identifying bursts, and blocking bots (Section 3.6). More generally, our work is an important contribution towards a comprehensive understanding of emerging low-volume application DDoS attacks.

CHAPTER 4
MEMCA: MEMORY ATTACKS ON THE NEIGHBOR'S CPU

In this chapter, we investigated the feasibility of resource contention and performance interference to locate a target VM (virtual machine) and degrade its performance. We explored external Very Short Intermittent DDoS Attacks by modeling the n-tier Web applications with queuing network theory and implementing the attacking framework based-on feed-back control theory. Through MemCA attacks on the RUBBoS benchmark web applications in our private cloud, we confirmed that the emerging elastic Cloud Computing can not defend against ever-evolving new types of DDoS attacks, since they exploit various newly discovered system or vulnerabilities even in the cloud platform, bypassing not only the elasticity mechanisms of Cloud Computing but also the state-of-the-art detection mechanisms of performance interference.

4.1 Introduction

Distributed Denial-of-Service (DDoS) attacks for web applications such as e-commerce are increasing in size, scale and frequency [1,2]. On October 21, 2016, A number of popular websites, like Twitter and Spotify, can not be accessed by some users worldwide, due to two DDoS attacks on the DNS servers of Dyn company [79]. On the other battlefield, Berkeley's paper "above the clouds: a Berkeley view of cloud computing" [3] first introduced Cloud Computing in 2009, they predicted the top one opportunity of Cloud Computing is that it can use elasticity to defend against DDOS attacks due to its easily scaling for fitting the dynamic user requirements, even serving the attack traffic. Until today, Cloud Computing has a rapid development and been widely adopted. A large number of web-sites are moved onto the cloud [4,5,80](e.g., Spotify [81] moved its core infrastructure to Google Cloud on Feb. 23, 2016). However, DDoS attacks are still very active and even more severe, since ever-evolving new types of DDoS attacks that exploit various newly discovered network or system vulnerabilities even in the cloud, bypassing not only the state-of-the-art defense mechanisms [6,7] but also the elasticity mechanisms of Cloud Computing [59,82,83].

Traditional Denial Of service (DoS) attacks hurt the availability of the target service by overloading its resources [6,7,24]. In the cloud environments, the attacking approaches are much more flexible and abundant, can be categorized in two classes: external and internal attacks [25,26]. External attacks are the most orthodox form of DoS [6,7,24]. For example, VSI-DDoS attacks and Tail attacks [59,82] exploit the external HTTP requests supported by the victim web-sites deployed in the cloud to create transiant resource bottleneck of the target Web system and hurt the performance of service, such as long-tail latency problem. In contrast, internal attacks are new-born, which are emerging simultaneously with Cloud Computing by virtualization technique [25, 26]. Internal attacks mount the adversarial programs in the co-located VMs [27] (on the same host with the target VM) can cause resource contention and performance interference of the target VM and hurt its performance of service [12,29].

Existing solutions detect and mitigate the performance interference by provider-centric [84, 85] and user-centric [11, 13, 86] approaches. For the provider-centric approach, the cloud providers profile the infrastructure-level metric from the hosts in the cloud. Due to the consideration of a worthwhile investment of the cloud providers (e.g., profiling overhead should under 1% [87]), the cloud providers typically adopt coarse granularity monitoring (long-lived interference detection with several minutes). For example, the sampling intervals of Amazon's monitoring tool CloudWatch [48] and Microsoft Azure Application Insights [65] are both 1 minute, which are used as the trigger metrics for the cloud scaling. In the chapter [84], the default sampling duration is 10 seconds and sampling frequency is every 1 minute, in this configuration they can detect the interference of several minutes. For the user-centric approach, the cloud users pretect themselves from performance interference from their rental VMs (Vitrual Machine) in the cloud. The cloud tenant can sacrifice some overhead to detect the short-lived (less than a minute) performance interference using more less sampling interval (e.g., 5 seconds [13, 86]), however, a smaller sampling interval may incur higher noises (higher false positive) [86].

In this chapter we present a new type of very-short-lived internal attacks inside the cloud, which can significantly hurt the performance of web applications in the cloud (such as contratual violation) while bypassing the elasticity mechanism and the interference detection mechanism in the cloud. In the scenario of the proposed attacks, an attacker mounts an adversarial program in the co-located VMs with the target VM, which deploys the target web system; it can degrade the performance of the latency-sensitive web systems through creating very-short-lived (less than 5 seconds) performance interference among the co-located VMs hosted in the same physical machine in the cloud, which is a severe threat for the websites requiring the optimal performance and responsiveness, such as e-commerce, media streaming or online gaming. At the same time, the performance interference caused by the proposed attacks last in very-short-lived periods (less than 5 seconds), from the Sampling theory, the average system resource utilization ratio can be in the moderate level using coarse granularity monitoring, not only avoiding the triggering condition of the cloud scaling but also escaping the state-of-the-art dection mechanisms of performance interference in the cloud.

To achieve this attack end, the primary challenging task is to choose the target attack resource. In cloud environments, the shared hardware and software resources among the co-located VMs are essential for internal attacks, such as network bandwidth [88,89], I/O [90], last level cache [29], memory lock [29], CPU scheduling mechanism of the hypervisors [91], even cross-resource contention([92] measures the impact of network to last level cache). Those cross-resouce interactions [93] make it more difficult to improve resource efficiency in the cloud, the same as to detect performance interference due to the increasing difficulty for indentifying the source of interference. Thus, we choose the total on-chip hardware resource as the target attack resource. We explore a representative type of internal attacks, which intermittently triggers the memory bus contention among the co-located VMs, finally transiently saturates the CPU of the target VM, we name it *MemCA* in this chapter (Memory Attacks on the Neighbor's CPU).

In brief, this work makes the following contributions:

- We investigate a type of cross-resouce contention that can significantly cause performance interference among the co-located VMs, which can be used to locate a co-located malicious VMs (virtual machine) with the target VM and degrade its performance.

- We present a type of external Very Short Intermittent DDoS Attacks, MemCA attacks, that leads to the long-tail latency problem (or SLA violations) in web applications while the average resource utilization of the target servers are far from saturation.

- We model the impact of our attacks in n-tier systems based-on queuing network theory, and based-on the proposed model, we implement a feed-back control attack framework that dynamically tune the optimal attack parameters to fit the dynamics of system resource state.

- We validate the practicality of our attacks through attack experimental results of the benchmark website in our private cloud, confirming not only the damage impact (long-tail latency issue or SLA violations) but also the stealthiness (bypassing the elasticity mechanisms of Cloud Computing and the detection mechanisms of performance interference).

We outline the remainder of this chapter as follows. Section 4.2 introduces the motivated background and states the proposed attack and its assumptions. Section 4.3 exploits the potential cross-resouce contentions in cloud platforms with all the state-of-the-art hypervisors and profiles the impacts of memory bus attacks in our RUBBoS environments. Section 4.4 designs a type of internal Very Short Intermittent DDoS Attacks (MemCA attacks), models our attack scenarios in an n-tier system using queuing network theory and analyzes the corresponding impacts caused by our attacks, implements a feed-back control

attack framework adjusting the optimal attack parameters to fit the unknowning black-box target systems. Section 4.5 shows our attack results of RUBBoS [30] benchmark website in our private cloud, which further confirm the damage and stealthiness of the proposed attacks, and the practicality of the control framework in more practical Web environments deployed in the Cloud. Section 4.6 presents the related work and Section 4.7 concludes the paper.

4.2 Background

4.2.1 Motivations

Infrastructure-as-a-Service (IaaS) cloud computing providers, such as Amazon EC2 [94] and Microsoft Azure [95], rent computing resources encapsulated as an instance (e.g., a guest Virtual Machine) to their clients. They define the computing resources of the instance with "coarse grained" granularity, comprising CPU, memory, storage, and networking capacity. This coarse definition of instance is simple and user-friendly to the end-users, however, technically the providers hide the performance risk [11,12] for the users' payment, even the leakage of the private information [27, 96]. The fundamental reason is the ubiquitous phenomenon of performance interference between the co-located instances (in the same host) inside the cloud [13, 85, 86].

Although virtualization technique provides a high level of isolated environment making a multi-tenant application possible, the level of isolation is far from adequate for the performance requirement of Service Level Agreement, remaining the observations of performance interference between the co-located VMs [13,29,86]. This significant performance interference issue in the cloud, blocking the promotion of cloud computing, is due to the two following main reasons:

1. *None-partitionable shared resource:* The microarchitecture of physical processors in Amazon EC2 [94] and Microsoft Azure [97] (e.g., Intel's "Haswell" processors) to a large extent uses shared last level cache, memory controller, and memory bus (bandwidth) by all the physical cores. These shared on-chip hardware resource is hard to

partitioned by virtualization technique. For example, Last Level Cache (LLC) [98] are shared among all the CPUs, and inclusive, which means all data cached in the L1 and L2 cache must be cached in LLC, and once the data are removed from LLC, they are also removed from all the low-level caches. This loose sharing mechanism increases the risk of LLC cache miss and causes performance inference of co-located VMs in the same host, especially for the cache- and memory-sensitive applications [85].

2. *Resource overbooking:* To maximum the revenue for the cloud providers, except for sharing hardware resources, they sometimes overbook their servers for their tenants. [22] claims a vCPU for Amazon EC2 small instances only can use at most 40% of a physical core due to overbooking the physical cores.

In the same processor package, except for core-private L1 and L2 caches, last level cache and memory scheduling components (e.g., memory controller bus, bank scheduler, channel scheduler) are all shared by co-located VMs (see Figure 4.1). The commercial cloud providers can assign the Memory size and CPU cores to specific instance for VM [94,95], but it is insufficient to isolate all the other on-chip memory resources, leading to significant resource contention between co-located VMs, such as memory bandwidth, even cross-resource contention or bottleneck shift. For example, RFA attacks investigate performance degradation caused by Net on LLC [92]; LLC contention causes both co-located VMs to require more memory bandwidth creating a memory bandwidth bottleneck; memory thrashing due to increasing cache miss in memory systems press CPU's scheduling, making CPU contention for application requests.

In this work, we will focus our investigation on a type of cross-resource contention, specifically, shared on-chip memory resource contention on CPU of the co-located VMs, *MemCA* attacks, and leave exploration of other types of resource contentions to future work.

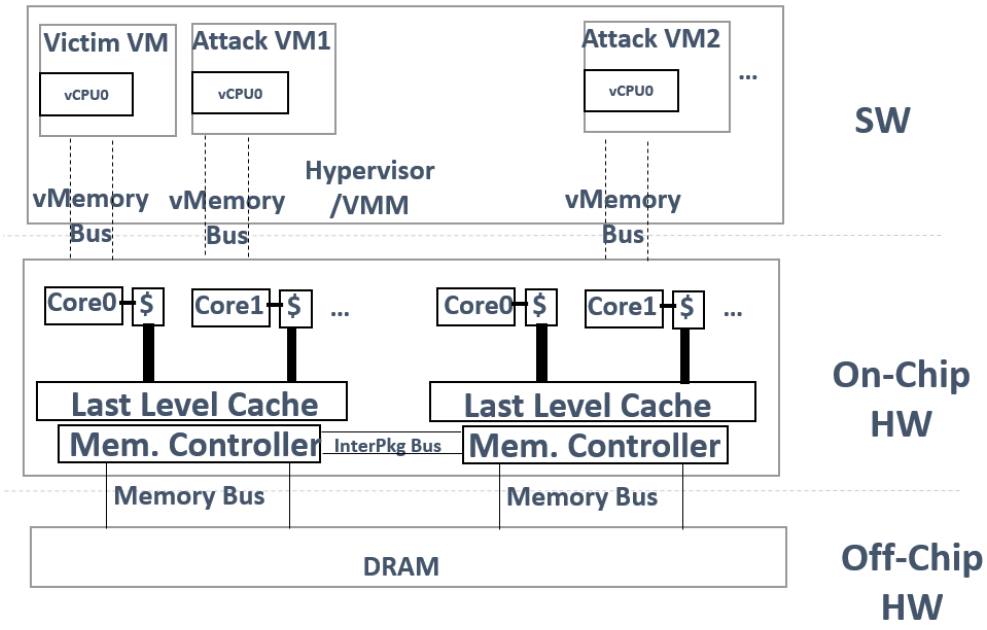


Figure 4.1. Shared on-chip resources including last level cache and memory scheduling components (e.g., memory controller bus, bank scheduler, channel scheduler).

4.2.2 Threat Model and Assumptions

We consider a MemCA attack scenario, in which the adversary frequently creates shared On-Chip memory resource contentions of the victim VMs in the cloud deploying the target web system by intermittently saturating those shared On-Chip memory resources (e.g., last level cache, buses, memory channels) without being detected. Today's software-based VMM (e.g., Xen, KVM, VMware vSphere, Microsoft Hyper-V) can only guarantee safe accesses to virtual and physical memory pages, but not to isolate those On-Chip memory resources shared by the VMs. In this case, the MemCA attacks can increase CPU consumption of the target VM by the memory attacks in the co-located adversarial VMs, even though vCPUs are isolated and protected by the hypervisor. Finally, such attacks is to degrade the quality of service for the target Web systems, especially causing long-tail latency problem (SLO violation) in the long run.

To effectively launch a MemCA attack, one precondition is the adversarial VMs should be shared the same host with the target VMs. We assume that the adversary can launch

at least one VM on the cloud hosts on which the target VMs are running. Many previous research efforts already provide solutions [27, 80, 99], and orthogonal to our research. Our attacks are launched in the VMs in the cloud, so we assume that the attacker can fully control the rented adversarial VMs, in other words, they can run any attacking programs in those VMs [100], as is the case for today's public IaaS cloud.

4.3 Memory Contention Measurements

4.3.1 Memory Bus Contention Measurements

In our testbed, we use three servers building up our private cloud, each one equips with a 12-core, 2-package 1.60 GHz Intel Xeon CPU E5-2603 v3 with 15MB of shared L3 cache per package and 16GB of main memory, a type of Physical Processors of Intel Xeon family, which is widely used to host their instances by Amazon EC2 [94]. This private cloud is managed by Openstack Ocata; each VM, running CentOS release 6.7, is managed by the hypervisor KVM [101].

The adversarial program to saturate memory bandwidth of the host, RAMspeed, is a cache and memory benchmarking tool [102]. RAMspeed supports multi processes which can trigger much more workload to memory bus. Due to having 6 cores per package in our host, we deploy 6 VMs, each with one vCPU. Thus, the attacking program run in each VM only with one thread. To comprehensively understand the impact of the chosen attacking program to the sharing memory bandwidth, we execute the program with one thread in three scenarios:

1. Same core, 6 VMs sharing all levels of processor cache, [22] focuses on this case to analyze the significant contention.

2. Same package, 6 VMs each pinned to a separate core on the same package, which share the last-level cache and memory bandwidth per package.

3. Different package, 6 VMs floating over 12 cores on two packages, which share the last-level cache and memory bandwidth of both packages.

When RAMspeed runs with one thread, it can saturate the CPU usage. Table 4.1 shows memory bandwidth of multi VMs measured by RAMspeed with one thread. 1) When all the VMs run on the same core, the attacking program has a maximum consumption of memory bandwidth in spite of the number of VMs (e.g., approximately 6700), since all the VMs share memory bus for the same physical core. Thus, one adversarial VM maybe not enough to saturate memory bus as large proportion as possible. 2) In the same package case, each VM shares the on-chip memory resource (e.g., Last level cache, and the control Bus). As the number of co-located VMs increase, the used memory bandwidth for each VM decreases accordingly, due to their competition for the same memory bus per package. 3) For the different package case, the available memory bus should double than the same package case. Thus, each VM can reach more high memory bandwidth than one in the same package. When there are 2VMs running on the different core of the different package, the bandwidth of each one also reach maximum value the attacking program can trigger.

Table 4.1. Measured used memory bandwidth with multi VMs, RAMspeed specifies 1 thread to execute in each VM. MemBW: Memory Bandwidth (MB/s).

Case	Co-located VMs (#)	1	2	3	4	5	6
SameCore	Used MemBW/VM	6747	3357	2227	1677	1332	1119
SameCore	Total Used MemBW	6747	6714	6681	6709	6662	6718
SamePkg	Used MemBW/VM	6769	4008	2724	1988	1605	1343
SamePkg	Total Used MemBW	6769	8017	8173	7952	8027	8063
DiffPkg	Used MemBW/VM	6596	6667	5097	3864	3232	2709
DiffPkg	Total Used MemBW	6596	13334	15291	15456	16160	16254

Except for the case of scaling out our attacking VMs (increasing the number of VMs), we also investigate the case of scaling up our attacking VMs (increasing the number of vCPU for one attacking VM). Table 4.2 shows memory bandwidth of one VM with 2 vCPUs measured by RAMspeed with multi-thread cases. When threads of RAMspeed outnumber vCPUS of VM, the effective fractions of the attacking program can not occupy all the CPU time, although the CPU is saturated during the execution. In other words, almost half of the CPU cycles is used to thrash in the kernel rather than increase the

pressure to memory bus in the user space. The takeaway is that the number of threads of RAMspeed should equal the number of vCPUS of VM to maximum the effectiveness of RAMspeed. For one thread case, the memory bandwidth in the different package is the worst, due to the inter-package bus is much slower for the communication between the two vCPUs. The best scenario is the two attacking threads and two vCPUs in the same package, they can reach enough memory bandwidth during the least executing time, which implies that it can incur the most pressure of memory bus per package. Although the same level of memory bandwidth can measured in the different package case, higher capacity of memory bus in the different package can disperse the contention of memory bus caused by RAMspeed.

Table 4.2. Measured used memory bandwidth in a VM with 2 vCPUs, RAMspeed runs with multi threads. MemBW: Used Memory Bandwidth (MB/s).

Case-#Threads	MemBW(MB/s)	CPU job got(%)	ExecuteTime(s)	UserTime(s)
SameCore-1	6490	99	1.8	1.6
SameCore-2	6594	99	3.5	3.2
SameCore-4	7490	59	7.2	3.3
SamePkg-1	6688	99	1.7	1.6
SamePkg-2	8199	98	2.8	2.6
SamePkg-4	9051	48	5.5	2.6
DiffPkg-1	4327	99	2.6	2.4
DiffPkg-2	8253	99	3.4	3.0
DiffPkg-4	8335	51	7.0	3.4

We profile the capacity of servers hosting VMs and pressure level caused by the adversarial program. From the previous experiments we can conclude that

1. One VM or one thread running the attacking program can not trigger the contention of memory bus, since the bus bandwidth in modern processors may be too high.

2. The biggest impact of memory bus contention is the VMs scheduled in same core, next in same package, last in different package. This implication is that memory bus can be split into three cases, including in same core, in same package, in different package. The capacity for each one increases from top to bottom, as shown in Figure 4.1.

3. The adversarial program by increasing the VMs or vCPUs can effectively dial up the pressure of memory bus. For multi-vCPUs approach, the number of threads of the attacking program should equal the number of vCPUs of the VM. For multi-VMs approach, the attackers should coordinate and synchronize the adversarial program in multi-VMs.

4. The most effective attacking condition is the used memory bandwidth caused by the attacking program is significantly high while keeping the executing time as short as possible. Only in this case, our attacks can create an effective and stealthy inference burst. Due to the solutions for long-lived [84, 85](minutes) and short-lived [13, 86](seconds) interference detection, our goal is to create very-short-lived inference burst (less than 5 seconds, even sub-second).

4.3.2 Memory Bus Contention Measurements in Other VMMs

The previous Memory Bus Contention experiments are conducted in KVM platform, in order to further confirm that these potential memory bandwidth attacks in the co-located VMs do not concern the hypervisors and can apply to all the Cloud platform nowadays. We further conduct the similar experiments in different platform managed by all the popular hypervisors. Table 4.3 lists all the IaaS vendors and its used hypervors. Table 4.4 lists measured total used memory bandwidth with multi VMs in various hypervisors in the same package case, and RAMspeed specifies one thread to run. We can not control the placement of VMs in Microsoft Azure, so we do not show the results. However, we can observe the maximum available memory bandwidth of the Azure instances is at same level (around 3000 MB/s).

4.3.3 Memory Bus Attacks on CPU in RUBBoS

To investigate Memory Bus Attacks on CPU in Clouds, we adopt RUBBoS [30], a representative n-tier web application benchmark modeled after the popular news website *Slashdot* [103]. We use a typical 3-tier configuration, with 1 Apache web server, 1 Tomcat application Server, and 1 MySQL database server. Figure 4.2 shows RUBBoS sample

Table 4.3. IaaS Clouds and their Hypervisors. We will profile memory bus contentions in all of these Hypervisors.

Hypervisors	IaaS Clouds
Hyper-V	Microsoft Azure
Xen	Amazon AWS, M5cloud, Rackspace, Storm, Lecloud, Gigenet, eApps, NSF-Cloud
KVM	Google Compute Engine, Atlantic, E24cloud, Exoscale, Cloudsigma, Cloudware, Joyent, Vpsnet, Elastichosts
VMware	Bitrefinery, Stratogen, Zettagrid, Ddata, Hyve

Table 4.4. Measured total used memory bandwidth (MB/s) with multi VMs in various Hypervisors in the same package case (except Azure), RAMspeed specifies one thread to run.

Clouds	CPU Freq	VMMs	1VM	2VMs	3VMs
Private	1.60GHz	KVM	6769	8017	8173
Private	1.60GHz	VMware	6857	7725	7803
NSF	2.10GHz	Xen	8942	11008	10370

topology in our controlled private cloud. RUBBoS has a workload generator to emulate the behavior of legitimate users who generate HTTP requests to interact with the target benchmark website. Each user has an average 7-second think time between receiving a web page and submitting a new page request. RUBBoS supports 24 different web interactions, we use CPU-intensive transactions to emulate the legitimate users, since our attack goal is to trigger the CPU bottleneck of the target VM. We choose four level workloads from low to high by configuring 1000, 2000, 3000, 4000 concurrent legitimate users, the corresponding CPU utilization of MySQL from 23% to %72. We deploy our 5 attacking VMs co-located with MySQL VM, and pin all the 6 VMs in the same package for simplicity. The bad impact of the same core case is easy to trigger and investigated [22], the different package case can be extended only by increases more VMs or vCPUs. Our attacking goal is to cause the SLO violation (e.g., 99ile response time over hundreds of milli-seconds), we gradually increase the number of co-located attacking VMs until the attack goal is achieved.

Table 4.5 lists the corresponding response time and the CPU utilization of MySQL under different workloads and attack intensity. When we run RAMspeed in the adversarial

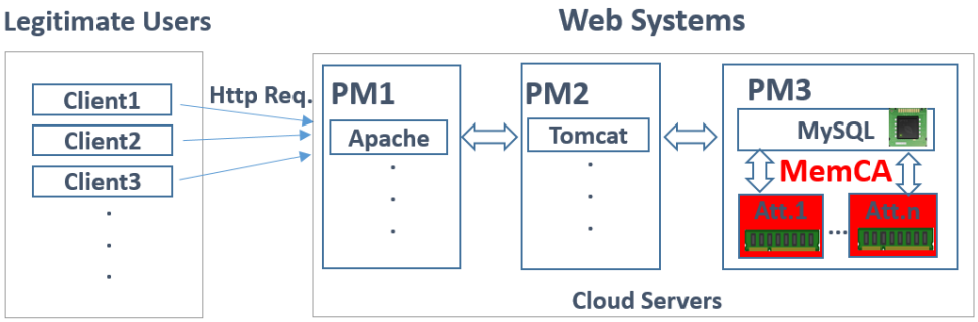


Figure 4.2. Experimental sample topology in our controlled private cloud. The adversarial VMs are co-located with MySQL VM.

VM, to achieve the SLO of the latency-intensive workloads (e.g., 99ile response time within tens of milli-seconds [93]), the CPU utilization of the target VM MySQL increases as the number of attack VMs increases. Once CPU utilization of the target VM MySQL is over 80%, SLO violation occurs (the yellow cell in the table). If we continue to increase the contention, CPU utilization of MySQL is saturated, causing more serious performance problem (the red cell in the table), which is like the flooding DDoS attacks out of our research. We only focus on how to effectively trigger SLO violation while keeping our attacks much more stealthy. In additional, it is obvious that to achieve the goal of SLO violation less VMs are required as the workload increases, the reason is that higher workload requires more working set of cache and consumes more memory bus, thus less VMs can trigger effective memory contention.

Table 4.5 also lists the user space CPU utilization of MySQL under different workloads and attacking intensity. User space CPU utilization implies the amount of time the CPU was busy executing MySQL daemon code in user space, except its execution in kernel space (e.g., a system call by MySQL daemon). High user space CPU utilization means that MySQL daemon must handle more requests during the contention (throughput decreases and more requests are queued in MySQL daemon), the fundamental reason is the memory bandwidth contention or LLC contention [11] triggered by our adversarial program, leading to more CPU time to service the cache misses.

Table 4.5. Observations of response time and potential bottleneck resource (MySQL CPU utilization) in RUBBoS environments.

Workload-AttackVMs	avgRT(ms)	99ile	98ile	95ile(ms)	CPU	UserCPU(%)
1K-0VMs	11	32	29	24	23	15
1K-1VMs	12	34	30	25	28	21
1K-2VMs	12	40	35	29	39	30
1K-3VMs	14	60	48	37	51	39
1K-4VMs	16	77	62	47	65	49
1K-5VMs	43	259	206	143	79	55
2K-0VMs	11	35	31	25	40	28
2K-1VMs	12	40	35	28	49	35
2K-2VMs	13	57	47	35	62	46
2K-3VMs	18	81	62	38	78	60
2K-4VMs	22	145	116	81	85	61
2K-5VMs	1738	15740	9673	5928	92	68
3K-0VMs	11	42	35	27	56	38
3K-1VMs	13	58	48	35	66	48
3K-2VMs	17	113	90	62	84	60
3K-3VMs	482	3813	2770	1612	98	70
4K-0VMs	13	64	51	35	72	46
4K-1VMs	17	111	87	57	84	59
4K-2VMs	674	6321	3772	2429	99	70

4.4 MemCA Attacks

4.4.1 MemCA Overview

Through the experiments in Section 4.3, we profile the capacity of the target host and the attack force of the adversarial program. thus it is feasible to propose the conceptual framework to make our attacks efficiently and stealthy. To efficiently launch a MemCA attack while keeping stealthy, the most effective attacking approach is to saturate the shared source (e.g., the memory bus) quickly in miliseconds or seconds level to bypass the detection [13, 84, 86], and intermittently repeat this attacking process to cause the long threat of performance. In other words, our attacks should create very short intermittent resource contention bursts. To this end, we formally propose MemCA attacks as follows (Figure 4.3):

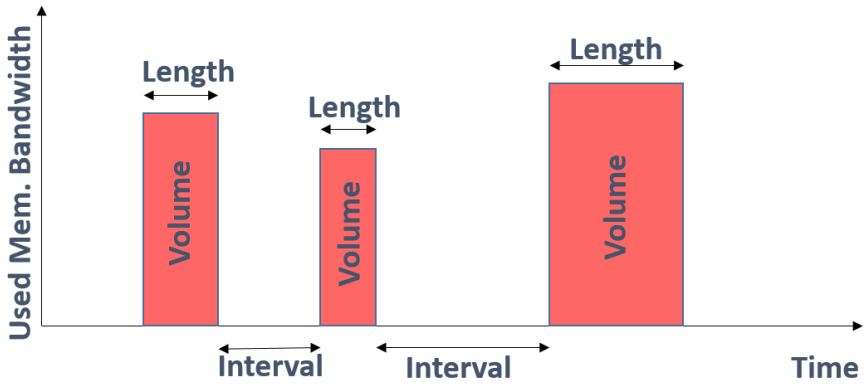


Figure 4.3. An illustration of a MemCA attack, which consists of interference burst volume V, length L, and interval I.

$$Effect = \mathbb{A}(V, L, I) \tag{4.1}$$

where,

- $Effect$ is the measure of attacking effectiveness by SLO requirements; we use percentile response time as a metric to measure the tail latency of the target system (e.g., 99ile response time $>$ 100ms). $Effect$ is a function of V, L, I.
- V is the amount (volume) of resource consumption per interference burst. V should be large enough to temporarily trigger the sharing resource contention (e.g., the memory bus) in the servers of the cloud.
- L is the period of each interference burst. L should be short enough to guarantee the burst not to be captured by the interference detection mechanisms [13, 84, 86].
- I is the time interval between every two consecutive interference bursts. I infers the frequency of bursts of our attacks. I should be short enough so that the attacker can generate interference bursts frequent enough to cause significant performance damage on the target system. On the other hand, too short I makes the attack similar to the traditional flooding DDoS attacks, which can be easily detected.

Given this high-level design of MemCA attacks (see Equation 4.1), two key challenges remain to make MemCA practical. They are addressed in the following Sections.

- How can we crystallize the relationship between attack impacts and attack parameters? Section 4.4.2 will generalize and formulate MemCA attacks' impacts using queuing network to model the n-tier systems.

- How can the attackers get the optimal attack parameters in the case of unknowing performance parameters among the n-tier system? Section 4.4.3 will exploit feedback control theory to dynamcally adjust attack parameters for better attack effectiveness and stealthiness, even unknowing performance parameters of the target systems.

4.4.2 MemCA Modeling

Queueing networks are typcially used to analyze performance problems and resource requirements in distributed systems [104], such as n-tier web applications [82]. In queueing theory, an M/M/1 queue (Kendall notation [105]) is usually exploited to represent one tier (a single server) with arrival rate of a Poisson process and service rate of an exponential distribution, such as the RUBBoS model [30]. M/M/1 queue is the fundamental unit to build up complex queueing networks. Figure 4.4 represents a queueing network of a classic n-tier web applications in our RUBBoS experiment (1 Apache, 1 Tomcat and 1 MySQL). We exploit the critical system resource of every tier to denote the corresponding tier, since typically major portion of the response time of each tier is occupied by the critical system resource of that tier [9]. For example, CPU is the bottleneck resource in MySQL, thus we use an M/M/1 queue waiting for CPU to depict MySQL; In Tomcat, there is a DB connection pool serving querying from MySQL, thus we use an M/M/c queue waiting for a DB connection denoting Tomcat, here, c denotes the idetential software resource (DB connection); In Apache, we use the Multi-Processing Module [106] that implements a hybrid multi-threaded multi-process web server, thus we also use an M/M/c queue waiting for a thread to denote Apache, here, c denotes a thread.

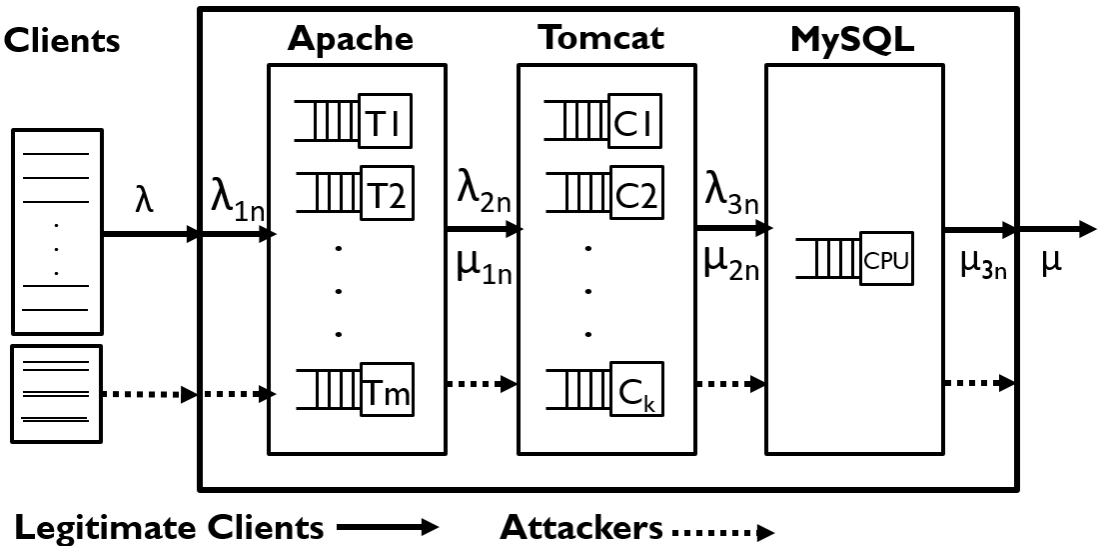


Figure 4.4. MemCA attack scenario and system model representing a queueing network of a classic 3-tier web applications in our RUBBoS experiment.

In our system model, we assume clients' arrival rate λ is a Poisson process, MySQL's service rate μ is an exponential distribution. We also assume μ is constant, which can imply the capacity of the server. In the n-tier system, we assume the bottleneck resource is MySQL CPU. One request going through to MySQL holds a thread of each upstream tier. If the queue of downstream tier is full, the incoming requests will be blocked in the upstream tier, which leads to the queue overflown propapation phenonema in n-tier systems [59, 82]. Due to the strong dependency among those distributed nodes, we can explicitly define the arrival rate and leave rate for each tier in the 3-tier web applications as follows, here, n is the queue length of each tier, m and k is the queue size of each tier,

$$\lambda_{1n} = \begin{cases} \lambda, & \text{for n} < \text{m}. \\ 0, & \text{for n} = \text{m}. \end{cases} \tag{4.2}$$

$$\lambda_{2n} = \mu_{1n} \tag{4.3}$$

$$\lambda_{3n} = \mu_{2n} \tag{4.4}$$

$$\mu_{1n} = \begin{cases} \mu_1, & \text{for n < m-1.} \\ \mu, & \text{for n = m-1.} \end{cases} \tag{4.5}$$

$$\mu_{2n} = \begin{cases} \mu_2, & \text{for n < k-1.} \\ \mu, & \text{for n = k-1.} \end{cases} \tag{4.6}$$

$$\mu_{3n} = \mu \tag{4.7}$$

Our final goal is to get the attack parameters to achieve the predefined attack goal (e.g., long tail latency). Based on the system model for n-tier systems, we should quantify the attack impact, in other words, solve the percentile response time in Equation 4.1. In queueing network theory, prior work [107] has suggested the performance analysis of the basic unit of an M/M/1 queue with a Poisson distrubition arrival rate λ and an exponential distrubition service rate μ, the rth percentile value of response time w is given by

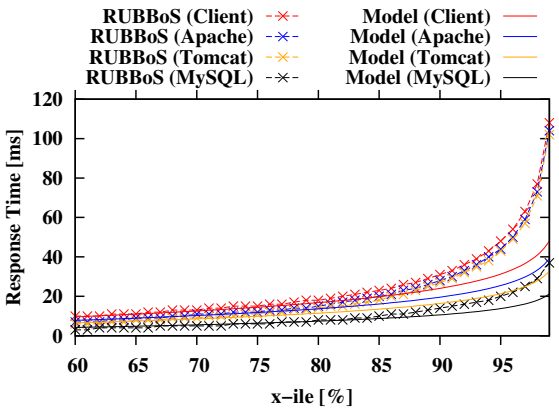
$$\pi_w[\text{r}] = W * ln(\frac{100}{100 - r}) \tag{4.8}$$

where, W is average response time of a request in the system with an M/M/1 queue.

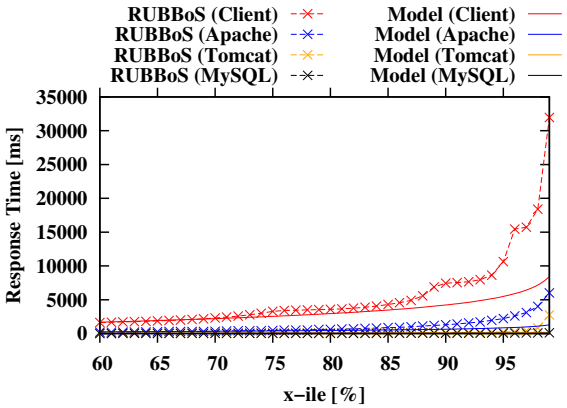
To solve the percentile response time in n-tier systems, we first analyze it from the last tier (MySQL), which is modelled by an M/M/1 queue, so we can exploit Equation 4.8 to solve the percentile response time of MySQL. Next, we combine the last tier (MySQL) and the second last tier (Tomcat) together to analyze the percentile response time of the second last tier. In our attack period, all the queue in the n-tier system is overflown. Thus, the service rate of the system is depend on the bottleneck tier, in our case, it is MySQL CPU (the system service rate is μ_{3n}). So we can treat Tomcat in our attack period as an

M/M/1 queue, apply Equation 4.8 to solve the percentile response time of Tomcat, W is average response time of a request residing in Tomcat, the start time is the request arriving Tomcat and the end time is the request served by MySQL and returning Apache (How to estimate the response time can refer to previous papers [59, 82]). Similarly, we can analyze the percentile response time of every upstream tiers until the most front tier. Finally, we get the percentile response time of the n-tier system.

Here, we present a set of RUBBoS experiments to validate the propsed model and analysis approach. Figure 4.5 depicts percentile response time of two MemCA Attacks cases in our RUBBoS experiments: low attack volume and high attack volume, and the curve line labelled "model" depicts the corresponding value calculated from Equation 4.8 using recursive solving approach mentioned in the previous section. The RUBBoS "tail" is more obvious than the model "tail". In M/M/1 model, the average/tail response time is inversely proportional to 1-order of the unused resource utilization [107]. In practice, the average/tail response time is inversely proportional to n-order of the unused resource utilization [108]. Thus, the RUBBoS tail increases faster and more steeply than the model tail. In additional, in high volume case, the tail of each tiers can be split, there exists gap of response time between two consecutive tiers at the tail, while in low volume case, the tails of Client, Apache and Tomcat are nearly overlapped. The reason is in high volume case, due to the impact of system bottleneck of MySQL CPU, more and more requests are queued in somewhere of the n-tier system, some are queued in MySQL waiting for MySQL CPU, some are queued in Tomcat waiting for DB connection, some are queued in Apache waiting for thread. Each request queued in upstream tier can be served only after some requests are released from the downstream tier. Thus, it has a large waiting time for each tier. Especially, for the Client tail, it increases dramatically, that's beacuse the dropped requests by Apache can lead to TCP retransmission (its Timeout expiration is in several seconds). However, in low volume case, there are few requests queued in the upstream tiers, so it has a small waiting time for those upsteam tiers.



(a) Low attack volume scenario



(b) High attack volume scenario

Figure 4.5. Percentil response time of MemCA attack experiments: Model and RUBBoS experimental results.

To keep our attack stealthy while effective, we choose the ON-OFF pattern to intermittently launch very short high volume attacks as Equation 4.1. Equation 4.8 only can be applied in the ON attack period, and during the OFF attack, the system returns normal status with low queued requests, we can consider that there exists no long tail latency in the OFF attack period. Thus, during a MemCA attack, the percentile response time of the n-tier system, $\hat{\pi}_w[\mathrm{r}]$, should be recalucated given by Equation 4.9, and we also substitute $\pi_w[\mathrm{r}]$ with Equation 4.8.

$$\hat{\pi}_w[\mathrm{r}] = (\frac{L}{L+I}) * \pi_w[\mathrm{r}] = (\frac{L}{L+I}) * W * ln(\frac{100}{100-r}) \tag{4.9}$$

In queue theory, the average response time of an application W is inversely proportional to the unused critical resource utilization [11,107], in MemCA attacks, we control the used critical resource utilization of the target VM, ρ, through tuning the attack volume V. Therefore, mathematically, $\hat{\pi}_w[\mathrm{r}] \sim W \sim 1/(1\text{-}\rho) \sim V$.

In addtional, we should control the interference length [109] of our attacks to guarantee our attacks escapce the detection of performance interference (typically based on several seconds monitoring) and invalidate the Cloud scaling mechanisms (based on 1 minute monitoring). To this end, we control L less than 5 seconds of each ON attack period, such that the average utizliation can be in moderate level from Sampling theory (such as 50%) far from the alert thread of dection tools and the trigger condition of scaling mechanisms.

4.4.3 MemCA Implementation

From the attacker's point of view, the target n-tier system is a black-box, it is hard to know performance parameters among the n-tier system accurately. Given the proposed model and analysis approaches, to get the optimal attack parameters, we should estimate the response time of the transactions supported by the target web applications. Previous works give us a lot of solutions to estimate the response time for the complex distributed system [9]. We also need to know the critical resource utilization of the target n-tier system, such as the thread pool of Apache, the DB connection pool of Tomcat, CPU of MySQL. However, it is impossible to get these white-box information. From the analysis based on the proposed model, we understand those critical resource has a positive relationship with the attack volume among the n-tier system. Therefore, we can exploit some advanced feedback control tools (such as Kalman filter [68]) to dynamcally tune the attack parameters to fit the unknowing system state and the shifty background workload [82].

To effectively launch and control MemCA attacks, we propose an attack framework including two parts: MemCA front end (MemCA-FE) and MemCA back end (MemCA-BE) (Figure 4.6). MemCA-FE creates co-located VMs with the target VM, executes the adversarial program in each VM and reports the consumption of the sharing resource.

MemCA-BE consists of two entities: the prober which periodically sends HTTP requests to the target Web systems and monitors the performance metrics of the target, and the commander which dynamically controls the attacking parameters of the attacking VMs based on QoS and resource metrics of the historical attacking bursts.

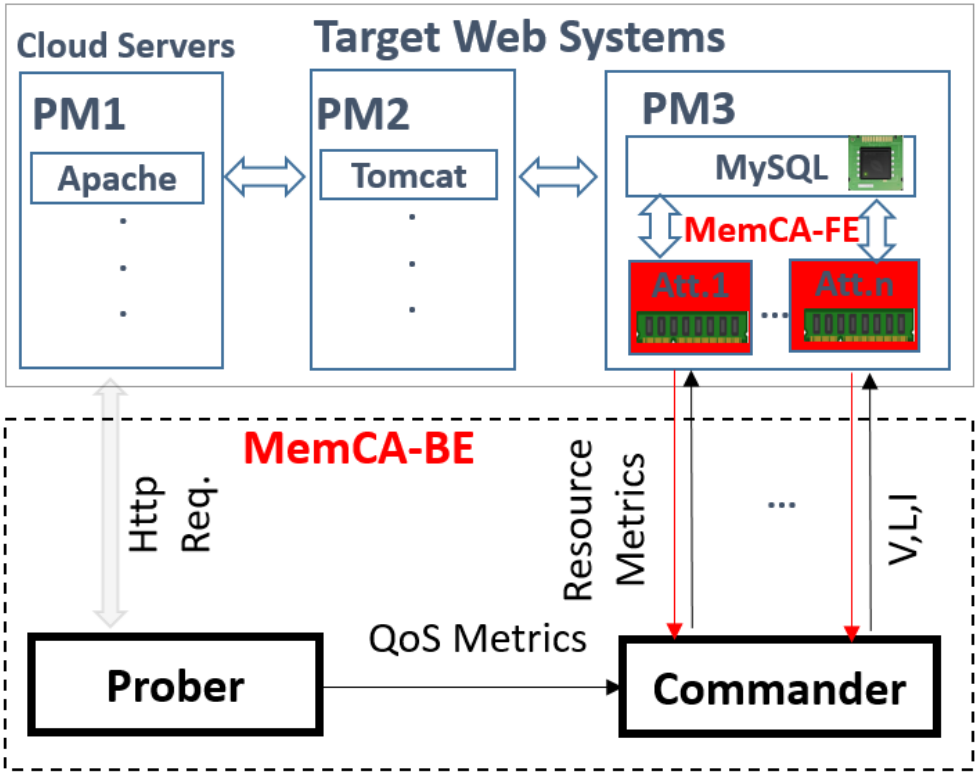


Figure 4.6. MemCA attack framework including two parts: MemCA attack front-end (MemCA-FE) and MemCA control back-end (MemCA-BE).

In MemCA-FE, we can record the critical resource utilization consumed by the attack VMs through the attack program (e.g., the RAMspeed benchmark). In our attack case, the critical resource is the memory bandwidth of physical machine hosting the attack VMs and the target VM. We also can profile the peak/maximum memory bandwidth in a specific hardware platform same as the target host machine. Finally, we normalize the resource usage ratio using the absolute value of measured memory bandwidth.

Here, we treat the n-tier system as a black-box M/M/1 queue, and measure the average and percentile response time of the transactions supported by the target system in the

prober of MemCA-BE. Due to the positive relationship between the response time and the resouce utilization, we can tune the attack volume V using feedback control technology, to control the target resource utilization and furture cause long tail latency problem or tail latency violations. Through adjusting the burst length L and the burst interval I, we can achieve a expected percentile response time of the target application (see Equation 4.9).

We also can record the execution time of the attack program (e.g., the RAMspeed benchmark) in the attack VMs in MemCA-FE. We assume the program can attack the bottleneck resource of the target host. During the execution time of the attack program, the target bottleneck resource is always busy and saturated. Thus, we can exploit the execution tiem as the interference length of the critical resource, further control the burst length L to bypass the state-of-the-art cloud detection of performanc interference. In practice, this approach is very conservative, during the execution time of the attack program there exist many tiny time slots the bottleneck resource can be idle. The actual attack can be more stealthy than the measured value using the execution time to estimate interference length.

4.5 MemCA Evaluation

Here, we use the RUBBoS benchmark web applications (the exprimental testbed and RUBBoS configurations refer to Section 4.3.1 and Section 4.3.3, respectively) to evaluate MemCA attacks from three aspects: the damage result and the stealthy results under cloud scaling and under cloud detection of performance interferrence.

4.5.1 MemCA in RUBBoS

In Section 4.3.3, we have shown the flooding memory bus attacks in RUBBoS (see red cells in Table 4.5). In that case, we do not adopt the ON-OFF attack pattern in Section 4.4.1, and we also do not exploit the feedback control to dynamically tune the optimal attack paratemters in Section 4.4.3. That's why CPU utilization of the target VM is always over than 90%, even full saturated. Here, based on the memory bus profiling of the target host in Section 4.3 and the model analysis and dynamically feedback control implementation in Section 4.4, we launch MemCA attacks in our private cloud environments to see the

attack impact. We fix the attack burst interval I as 10 seconds, and through the feedback control tools to adjust the optimal attack volume V and the burst length L. We predefine the attack goal is to cause the tail latency violations (such as 99% percentile response time bigger than 100 milli-seconds, which is Google's search Service Level Objective) and to bypass the state-of-the-art detection of performance interference (e.g., the interference length is less than 5 seconds).

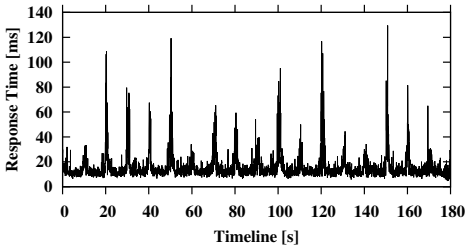
Figure 4.7 illustrates the 3-minute detailed experimental results with 3000 concurrent legitimate users under MemCA attacks launched by our control framework. We use two co-located VMs with the target VM (see Figure 4.2) to execute the attack program, these VMs are pinned in the same package in the same host in our private cloud, since two VMs are enough to saturate the memory bus in a host (see Table 4.1). Figure 4.7, a shows the point-in-time response time of the legitimate users under 3-minute MemCA attacks using 50ms granularity to caculate, the 99% percentile response time is bigger than 100 milli-seconds (not shown in the graph), the normal response time is around 10 seconds, and the peak response time is nearly 130 milli-seconds. We can see the periodic pulse of the response time every 10 seconds caused by the periodic ON-OFF MemCA attacks. Figure 4.7, c and 4.7, e show the corresponding attack volume and burst length recorded from the two attack VMs. The burst length are both around 2.5 seconds, far from the target performance interference length 5 seconds. Figure 4.7, d, 4.7, b, and 4.7, f zoom up an attack burst of 10 seconds (the burst occurs at the time point of 20 seconds), explaining the fundamental reason of the deterioration of response time. At the time point of 20 seconds, the two attack VMs simultaneously execute the attack program to occupy the memory bus of the target host. In this point, they will shrink the available memory bus for the co-located MySQL VM, so it leads to more cache miss in MySQL VM (it is low level metric from the cloud providers, which will be shown in Section 4.5.3). It causes the User CPU usage of MySQL VM increases as shown in Figure 4.7, b, due to cache miss. Finally, the total CPU usage of MySQL is saturated during the attack burst as shown in Figure 4.7, d. Such milli-

bottleneck in the n-tier system leads to long response time as shown in Figure 4.7, f [59,82]. In general, through the experimental results in the RUBBoS benchmark web application in our private cloud, we can confirm the damage and stealthiness impacts of MemCA attacks.

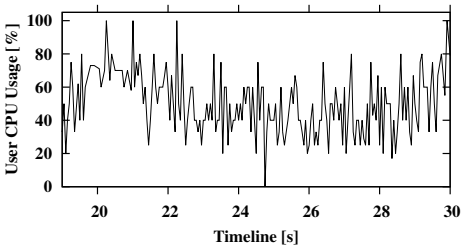
4.5.2 MemCA in Cloud Elasticity

Amazon supplies Elastic Load Balancing in its cloud platform AWS to guarantee scalability, performance, and security for any application [110]. It can elastically scale to the demands of the customers due to the ability to trigger Auto Scaling [46] in the available Amazon EC2 instance fleet. The trigger mechanism of Auto Scaling is based on coarse granularity monitoring tool, Amazon CloudWatch [48], which sampling period is 1 minute. For example, the customers' application can scale up more instances once the average CPU utilization of all the instances exceeds a preconfigured threshold (e.g., 85%) during a 1-minute sampling period. Thus, to validate whether Cloud Elasticity can mitigate our attack, it only requires to verify whether our attack will offend the threshold of Auto Scaling. Here, in our experiments we assume the preconfigured threshold of scaling up more instances is 85% of the average CPU utilization, which is usually a feasible and simple solution for system administrators [51, 52].

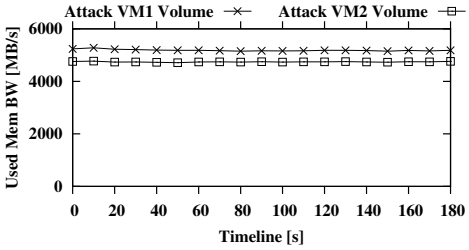
Figure 4.8 depicts CPU uitilization of the bottleneck tier MySQL in the previous 3-minute MemCA attacks experiments using different sampling granularity. Figure 4.8, a, 4.8, b, and 4.8, c respectively utilize 1-minute, 1-second, and 50-milli-second of monitoring granularity. The average utilization in 1-minute case is flat and moderate, it is obvious that it will not trigger the condition of Auto Scaling (e.g., Amazon Elastic Load Balancing). Using 1-second monitoring, it just shows a little bit fluctuation. Only using more fine-grained graunilairy 50 milli-seconds, we can observe the saturation of CPU uitilization. These peek value occurs frequently but transiently, thus the average value is at moderate level from Sampling theory as shown in Figure 4.8, a and 4.8, b . That's the essential reason why MemCA attacks can bypass the state-of-the-art cloud elasiticity mechanisms. Figure 4.8, d zooms up an attack burst of 10 seconds (the burst occurs at the time point of



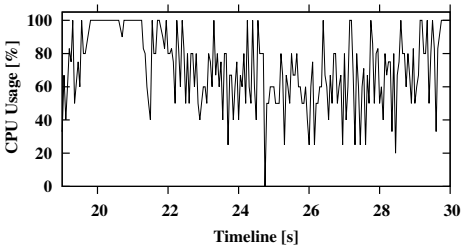
(a) Point-in-time response time of the legitimate users under 3-minute MemCA attacks using 50ms granularity. The peak response time is nearly 130 milli-seconds.



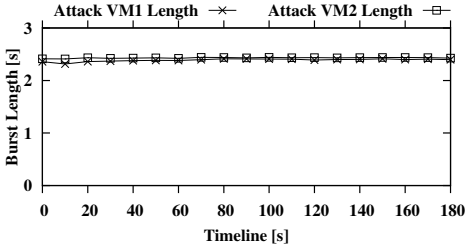
(b) User CPU usage of MySQL VM during an attack burst starting at the time point of 20 seconds. User CPU usage of MySQL VM reachs nearly 80% after the attack burst start.



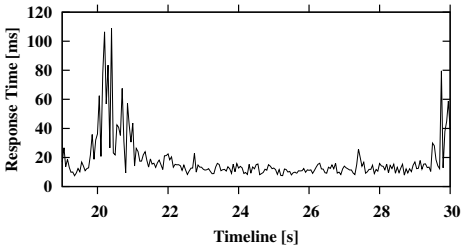
(c) The two attack VMs can consume all the memory bus of physical machine hosting the malicious VMs and the target VM.



(d) Total CPU usage of MySQL is saturated when User CPU usage of MySQL is in the peak period.



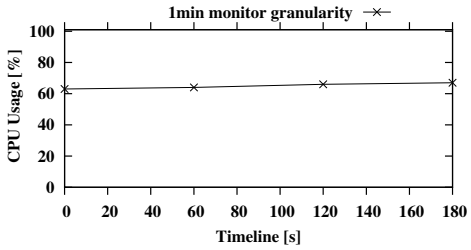
(e) Performance interference burst length of the attack programs executed in the two malicious VMs. The burst length are both around 2.5 seconds, far from the target performance interference length 5 seconds.



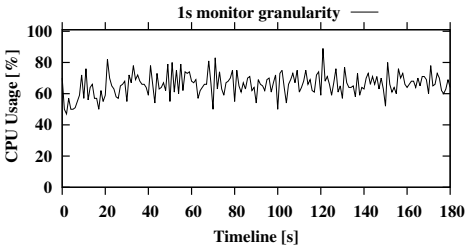
(f) Point-in-time response time of the legitimate users during the corresponding attack burst in Figure 4.7, d. Peak response time during ON attack burst is 5 times of normal response time during OFF attack burst.

Figure 4.7. MemCA in RUBBoS. Figure 4.7, a, 4.7, c, and 4.7, e show the performance impact (response time) caused by MemCA attacks during the 3-minute RUBBoS experiments. Figure 4.7, b, 4.7, d, and 4.7, f explain cause and result of MemCA attacks through zooming up an ON-OFF attack burst.

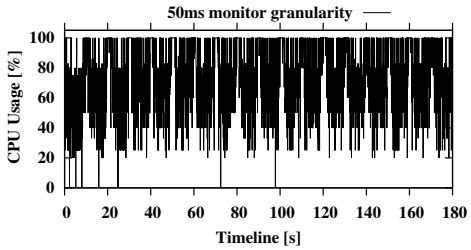
20 seconds), noting that the CPU saturation period last approximately 1 second. Through tuning the attacking paramters, we can control the saturation period to ensure the moderate average utilization of the target VMs.



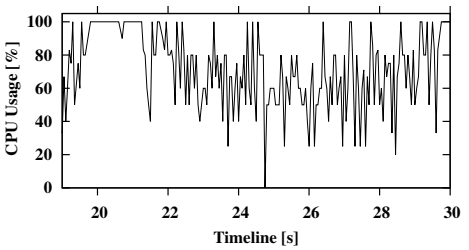
(a) Using 1min monitoring, CPU utilization can exceed the trigger condition of Auto Scaling.



(b) Using 1s monitoring just shows a little bit fluctuation.



(c) Using 50ms monitoring observes frequent CPU saturation pulsation.



(d) Zooming up a burst in Figure 4.8, c clearly shows transient CPU saturation.

Figure 4.8. MemCA in Cloud elasticity. The average utilization is at moderate level in Figure 4.8, a, not triggering the condition of Auto Scaling (e.g., CPU utilization is $> 85\%$.), even though the performance of the target applications experiences long response time as shown in Figure 4.7, a under MemCA attacks.

4.5.3 MemCA in Cloud Detection of Performance Interference

The sources of performance interference inside the cloud typcially are low-level metrics (e.g., hardware performance counts), which can not be measured for the customers of VMs, such as cache miss, memory bandwidth, etc. Thus, most detection approaches of performance interference are based on the low-level monitoring tools, such as Xentrace [111], OProfile [112]. Here, we use OProfile to measure LLC cache miss, since we try to intermittently saturate memory bus of the host in our private cloud, LLC cache miss is the most relevant hardware performance count with our attacks. We also use different monitoring

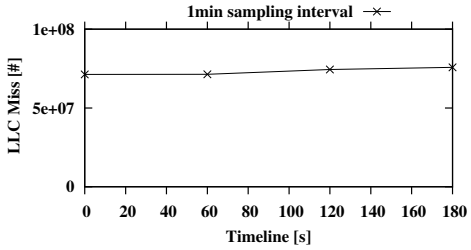
granularity to sample the chosen hardware performance count, similar with the observation of MySQL in Figure 4.8, we will show what's the difference among them for MemCA attacks.

Figure 4.9 shows LLC cache miss of physical machine hosting MySQL and its co-located VMs (PM3 in Figure 4.2) in the previous 3-minute MemCA attacks experiments using different sampling granularity. Figure 4.9, a, 4.9, b, and 4.9, c respectively use 1-minute, 10-second, and 1-second of monitoring granularity to record the amount of LLC cache miss. Since the attack interval of MemCA attacks in our experiments is 10 seconds, only using less than 10-second sample (e.g. 1-second) can trace the ON-OFF pattern of our attacks as shown in Figure 4.9, c. To further illustrate the attack pattern, we zoom up Figure 4.9, c to show LLC cache miss during a burst in Figure 4.9, d, noting the period of the peak is the length of performance interference, during which the end-user perceives the long response time as shown in Figure 4.7, f. To sum up, we confirm the same conclusion that the performance interference length is more important for the detection mechism [22], rather than the peak value. That's explain that why MemCA attacks model the interference length in Section 4.4.2 and estimate the interference length in Section 4.4.3, ensuring its high stealthy.

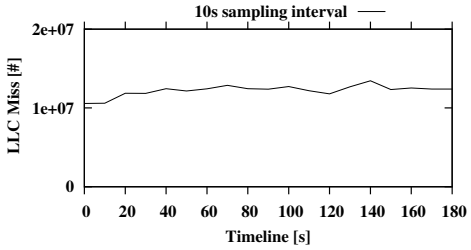
4.6 Related Work

In this section, we review the most relevant work with regard to memory performance attacks and cross-resource interference in the cloud.

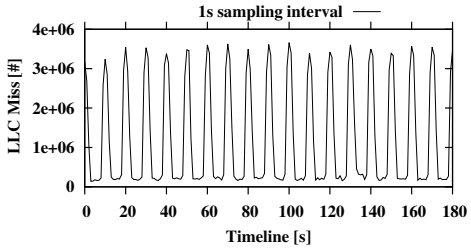
Memory is a critical bridge in modern computer architecture linking the high speed device (e.g., CPU) with low speed device (e.g.,disk), a good design hardware and software of using memory can smoothly blance the workload and siginificantly speedup the whole performance of the system. On the contrary, memory is also a most frequent computer resource exploited by the attacker to degrade the performance of the target system. The scenarios of memory performance attacks in multi-core systems [113] are too old to hold in modern hardward and cloud platforms. Zhang et al. [29] exploit two types of memory con-



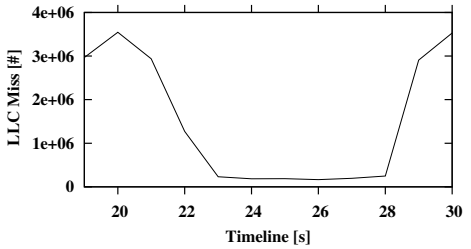
(a) Using 1min sampling inteval, LLC cache miss is flat.



(b) Using 10s sampling inteval, LLC cache miss is still flat.



(c) Using 1s sampling inteval, LLC cache miss shows obvious pulsation due to the ON-OFF MemCA attacks.



(d) Zooming up a burst shows the clear peak and valley of LLC cache miss during the ON and OFF attack period.

Figure 4.9. MemCA in Cloud detection of performance interference. Since the attack interval of MemCA attacks in our experiments is 10 seconds, only using less than 10-second monitoring granularity (e.g. 1-second as shown in Figure 4.9, c, 4.9, d) can trace the ON-OFF pattern of our attacks.

tentions, storage-based contention (LLC cleansing attack) and scheduling-based contention (exotic atomic locking attack), to degrade the performance of applications deployed in the cloud. Mehmet et al. [114] attack memory bus, focusing on the mobile device lauched by the attack APP. Compared to previous memory attacks, we investigated memory bus attacks, which can influence CPU of co-located VMs inside the cloud. This type of cross-resource attacks (the impact is CPU while the source is memory) is more hard to detect by the defender. More strikingly, the scenario of our attacks is on n-tier systems, in which there typically exists strong dependency among the distributed nodes [82], the damage of our attacks (long-tail latency) can be even stealthy and severe, due to tail amplification.

Mitigating a type of resource interference in the cloud can be feasible and easy to detect, widely investiged [84, 85, 115, 116]. Cross-resource interactions (e.g., MemCA attacks, the

impact is CPU while the source is memory) is a more difficult problem than single resource interference. Resource freeing attack [92] proposed one type of resource stealing attacks in the Cloud through cross-Resource interference, where a malicious VM increases its usage of network traffic to grab LLC from the co-located victim VMs. Heracles [93] integrated multiple isolation schemes to improve resource efficiency while satisfying latency critical workloads, but it requires a complex design since the number of possible interactions scales with the square of the number of interference sources due to cross-resource interferences. Finaly, after the continual efforts for the final goal in data-center (high utilization, low interference, and fast latency), Google's engineer has to admit that this is a tough and impossible task on the modern CPU architecture [117]. In fact, MemCA attack is a representative issue of high utilization, high interference and slow latency; mitigating MemCA attacks equals achieving the final goal for the cloud providers.

4.7 Conclusion

In this chapter, we described MemCA Attacks, a new type of external Very Short Intermittent DDoS Attacks [59], in which an attacker exploits a newly identified system vulnerability (resource contention in the colocated VMs inside the cloud) of n-tier web applications to cause the long-tail latency problem of the target web application with a higher level of stealthiness. We investigated a type of cross-resouce contention that can cause transient CPU saturations of the target VM through intermittently saturating the memory bus of the host by the co-located VMs, which can be used to locate a co-located malicious VMs (virtual machine) with the target VM and degrade its performance(Section 4.3). We modeled the n-tier systems using queuing network theory, and analyzed the damage and stealthy results of MemCA attacks. Based-on the proposed model, we implement a feedback control attack framework that dynamically tune the optimal attack parameters to fit the dynamics of system resource state(Section 4.4). To validate the practicality of our attacks, we evaluated our attacks through RUBBoS benchmark web applications in our private cloud (Section 4.5, and confirmed that our attacks can not only cause the damage

impact (long-tail latency issue or SLA violations) but also invalidate the elasticity mechanisms of Cloud Computing and the detection mechanisms of performance interference. In generally, MemCA Attacks is an important constituent part of Very Short Intermittent DDoS Attacks, the proposed attack in this dissertation.

CHAPTER 5
CONCLUSION AND FUTURE WORK

5.1 Conclusion

Despite of the benefits of cloud computing, many research efforts find the system vulnerability in the cloud in Chapter 1. I extend those works from the attacker's point of view, and exploit these existing system vulnerabilities and dig novel system holes to mount new types of attacks as an important complement for emerging DDoS attacks. We first present a new type of low volume application layer DDoS attacks, Very Short Intermittent DDoS (VSI-DDoS) Attacks , which can create transiently resource saturation of the target n-tier web system, causing long-tail latency (or SLA violations) while the average utilization of the target system is at moderate level guarantee its stealthiness.

In today's cloud enviornment, this attack can be catigorize into two types: external VSI-DDoS attacks and internal VSI-DDoS attacks. VSI-DDoS attacks (Chapter 2) and Tail attacks (Chapter 3) are two types of external VSI-DDoS attacks, which are launched by the external attack requests (heavy HTTP requests, refer to Section 2.4). Chapter 2 introduced the unque root of VSI-DDoS attacks, Very Short Bottleneck or Milli-bottleneck. To effectively launch external VSI-DDoS attacks, this chapter proposed an attack framework, which is based on the empirical approach, to get the attack parameters statically. Chapter 3 digged into the fundamental cause of external VSI-DDoS attacks, modeled the target system with queuing network thory and analyzed the damage impact with damage length and predicted the stealthiness level with milli-bottleneck length. To improve the attack framework in Chapter 2, we exploit a feedback control tool (e.g., Kalman filter) to implement a feedback control attack framework, which can dynamically tune the optimal attack parameters to fit the shifts of the background workload and the system state.

MemCA attacks (Chapter 4) is a type of internal VSI-DDoS attacks, which are launched by the adversarial attack program executed in the co-located VMs to cause the transiant resource contention with their co-located target VM. Chapter 4 introduced a type of cross-

resource internal VSI-DDoS attacks. The attacker can cause the transiant CPU saturation of the target VM through intermittently saturating the memory bus in the malicious VMs. We also modelled the target n-tier system with queuing network thory and analyzed the long-tail latency caused by internal VSI-DDoS attacks, and implement a feedback control internal attack framework.

Through the experimental results verifying the damage of a series of the proposed attacks (external attacks and internal attacks) in Chapter 2, 3, 4, we confirm that this work (the proposed VSI-DDoS attacks) can efficiently and stealthily degrade the performance of the target Web applications deployed in the cloud, causing long-tail latency issue (or SLA violations). We also show that our attack not only can bypass the state-of-the-art DDoS defend mechinsms, but also avoid the detection of performance interference in the cloud, even invalidate the cloud elasiticy mechinsms of auto-scaling.

Overall, through highlighting the practicality and impact of Very Short Intermittent DDoS Attacks, I safely expect that this work will motivate cloud providers to furnish much securer infrastructure to develop and deploy Web applications in the cloud.

5.2 Future Work

Very Short Intermittent DDoS Attacks aim at transiently saturating the critical bottleneck resource (e.g., database CPU, last level cache, memory bandwidth) of the target systems by external heavy requests outside the cloud or internal resource contention inside the cloud, which can saturate the critical resource of the system with much lower volume (thus less bots are needed) compared to that of traditional flooding DDoS attacks which usually try to fully saturate the target network bandwidth. In depth, we can dig into all sources of performance interference in the cloud platform to launch Very Short Intermittent DDoS Attacks, such as shared on-core resources (last level cache, memory bus), shared off-core resources (disk I/O, network bandwidth), global software resources (the scheduling algorithm of hypervisors) and other scheduled activity (periodic garbage collection in garbage-collected languages, overhead of monitoring or sampling tool).

In addition, we can evaluate Very Short Intermittent DDoS Attacks in different application dimensionality in the future. Except for the percentile latency, we can hurt the availability of data service in a "Very Short Intermittent" pattern in SaaS cloud platform, such as Dropbox, Google Drive, and Microsoft OneDrive. We also can investigate very short intermittent attacks for Web applications of various latency levels in different stealthy approaches, such as very low-latency services (e.g., search, content delivery) and a little longer latency tasks (e.g., online stream, video, recognition) run in today's elastic cloud platform. Additionally, emerging Internet of Thins (IoT) applications (e.g., Google Glass) in IoT cloud backends also can be the target of the proposed attacks. I expect that Very Short Intermittent DDoS Attacks is a type of advanced persistent threat in the long run, the notable obstacle blocking the rapid development and applications of cloud computing technology.

REFERENCES

[1] *Akamai QUARTERLY SECURITY REPORTS.* "https://www.akamai.com/us/en/about/our-thinking/state-of-the-internet-report/global-state-of-the-internet-security-ddos-attack-reports.jsp".

[2] C. Baraniuk, *DDoS: Website-crippling cyber-attacks to rise in 2016.* "http://www.bbc.com/news/technology-35376327/".

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica *et al.*, "Above the clouds: A berkeley view of cloud computing," Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Tech. Rep., 2009.

[4] K. He, A. Fisher, L. Wang, A. Gember, A. Akella, and T. Ristenpart, "Next stop, the cloud: Understanding modern web service deployment in ec2 and azure," in *Proceedings of the 2013 conference on Internet measurement conference.* ACM, 2013, pp. 177–190.

[5] L. Wang, A. Nappa, J. Caballero, T. Ristenpart, and A. Akella, "Whowas: A platform for measuring web deployments on iaas clouds," in *Proceedings of the 2014 Conference on Internet Measurement Conference.* ACM, 2014, pp. 101–114.

[6] J. Mirkovic and P. Reiher, "A taxonomy of ddos attack and ddos defense mechanisms," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.

[7] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks," *IEEE communications surveys and tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.

[8] Q. Wang, Y. Kanemasa, J. Li, C.-A. Lai, C.-A. Cho, Y. Nomura, and C. Pu, "Lightning in the cloud: A study of very short bottlenecks on n-tier web application performance," in *Proceedings of USENIX Conference on Timely Results in Operating Systems*, 2014.

[9] Q. Wang, Y. Kanemasa, J. Li, D. Jayasinghe, T. Shimizu, M. Matsubara, M. Kawaba, and C. Pu, "Detecting transient bottlenecks in n-tier applications through fine-grained analysis," in *Proceedings of 2013 IEEE 33rd International Conference on Distributed Computing Systems (ICDCS).* IEEE, 2013, pp. 31–40.

[10] Q. Wang, C.-A. Lai, Y. Kanemasa, S. Zhang, and C. Pu, "A study of long-tail latency in n-tier systems: Rpc vs. asynchronous invocations," in *Proceedings of 2017 IEEE 37rd International Conference on Distributed Computing Systems (ICDCS).* IEEE, 2017, pp. 207–217.

[11] S. A. Javadi and A. Gandhi, "Dial: Reducing tail latencies for cloud applications via dynamic interference-aware load balancing," in *Proceedings of 2017 IEEE International Conference on Autonomic Computing (ICAC).* IEEE, 2017, pp. 135–144.

[12] C. Delimitrou and C. Kozyrakis, "Bolt: I know what you did last summer... in the cloud," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2017, pp. 599–613.

[13] A. K. Maji, S. Mitra, B. Zhou, S. Bagchi, and A. Verma, "Mitigating interference in cloud services by middleware reconfiguration," in *Proceedings of the 15th International Middleware Conference*. ACM, 2014, pp. 277–288.

[14] C. Wang, B. Urgaonkar, A. Gupta, L. Y. Chen, R. Birke, and G. Kesidis, "Effective capacity modulation as an explicit control knob for public cloud profitability," in *Proceedings of 2016 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 2016, pp. 95–104.

[15] Y. Xu, M. Bailey, B. Noble, and F. Jahanian, "Small is better: Avoiding latency traps in virtualized data centers," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 7.

[16] X. Bu, J. Rao, and C.-z. Xu, "Interference and locality-aware task scheduling for mapreduce applications in virtual clusters," in *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*. ACM, 2013, pp. 227–238.

[17] R. Kohavi and R. Longbotham, "Online experiments: Lessons learned," *Computer*, vol. 40, no. 9, 2007.

[18] K. Curtis, P. Bodík, M. Armbrust, A. Fox, M. Franklin, M. Jordan, and D. Patterson, *Determining SLO Violations at Compile Time*, 2010.

[19] S. A. Baset, "Cloud slas: present and future," *ACM SIGOPS Operating Systems Review*, vol. 46, no. 2, pp. 57–66, 2012.

[20] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[21] M. Jeon, Y. He, H. Kim, S. Elnikety, S. Rixner, and A. L. Cox, "Tpc: Target-driven parallelism combining prediction and correction to reduce tail latency in interactive services," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2016, pp. 129–141.

[22] Y. Xu, Z. Musgrave, B. Noble, and M. Bailey, "Bobtail: Avoiding long tails in the cloud." in *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, vol. 13, 2013, pp. 329–342.

[23] J. Li, N. K. Sharma, D. R. Ports, and S. D. Gribble, "Tales of the tail: Hardware, os, and application-level sources of tail latency," in *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 2014, pp. 1–14.

[24] G. Mantas, N. Stakhanova, H. Gonzalez, H. H. Jazi, and A. A. Ghorbani, "Application-layer denial of service attacks: taxonomy and survey," *International Journal of Information and Computer Security*, vol. 7, no. 2-4, pp. 216–239, 2015.

[25] M. Darwish, A. Ouda, and L. F. Capretz, "Cloud-based ddos attacks and defenses," in *Proceedings of 2013 International Conference on Information Society (i-Society)*. IEEE, 2013, pp. 67–71.

[26] A. Bakshi and Y. B. Dujodwala, "Securing cloud from ddos attacks using intrusion detection system in virtual machine," in *Proceedings of Second International Conference on Communication Software and Networks (ICCSN'10)*. IEEE, 2010, pp. 260–264.

[27] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 199–212.

[28] J. Huang, D. M. Nicol, and R. H. Campbell, "Denial-of-service threat to hadoop/yarn clusters with multi-tenancy," in *Proceedings of 2014 IEEE International Congress on Big Data (BigData Congress)*. IEEE, 2014, pp. 48–55.

[29] T. Zhang, Y. Zhang, and R. B. Lee, "Dos attacks on your memory in cloud," in *Proceedings of the 12th ACM on Asia Conference on Computer and Communications Security (ASIA CCS'17)*. ACM, 2017, pp. 253–265.

[30] *RUBBoS*. "http://jmob.ow2.org/rubbos.html".

[31] *Kaspersky DDoS Intelligence Report for Q1 2017*. "https://usa.kaspersky.com/about/press-releases/2017_kaspersky-lab-report-on-ddos-attacks-in-q1-2017-the-lull-before-the-storm".

[32] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: Flexible and elastic ddos defense." in *Proceedings of the 24th USENIX Security Symposium*, 2015, pp. 817–832.

[33] M. Guirguis, A. Bestavros, and I. Matta, "Exploiting the transients of adaptation for roq attacks on internet resources," in *Proceedings of the 12th IEEE International Conference on Network Protocols*. IEEE, 2004, pp. 184–195.

[34] M. S. Kang, S. B. Lee, and V. D. Gligor, "The crossfire attack," in *Proceedings of 2013 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2013, pp. 127–141.

[35] Y.-M. Ke, C.-W. Chen, H.-C. Hsiao, A. Perrig, and V. Sekar, "Cicadas: Congesting the internet with coordinated and decentralized pulsating attacks," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. ACM, 2016, pp. 699–710.

[36] X. Luo and R. K. Chang, "On a new class of pulsing denial-of-service attacks and the defense," in *Proceedings of Network and Distributed System Security Symposium (NDSS)*, 2005.

[37] A. Kuzmanovic and E. W. Knightly, "Low-rate tcp-targeted denial of service attacks: the shrew vs. the mice and elephants," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*. ACM, 2003, pp. 75–86.

[38] A. Herzberg and H. Shulman, "Socket overloading for fun and cache-poisoning," in *Proceedings of the 29th Annual Computer Security Applications Conference*. ACM, 2013, pp. 189–198.

[39] IETF, *RFC 6298*. "https://tools.ietf.org/search/rfc6298/".

[40] Y. Zhang, Z. M. Mao, and J. Wang, "Low-rate tcp-targeted dos attack disrupts internet routing," in *Proceedings of Network and Distributed System Security Symposium (NDSS)*, 2007.

[41] P. Ramamurthy, V. Sekar, A. Akella, B. Krishnamurthy, and A. Shaikh, "Remote profiling of resource constraints of web servers using mini-flash crowds," in *Proceedings of the 2008 USENIX Annual Technical Conference*, 2008, pp. 185–198.

[42] S. Mathew, *Caching HTTP POST Requests and Responses*. "http://www.ebaytechblog.com/2012/08/20/caching-http-post-requests-and-responses".

[43] *Snort*. "https://www.snort.org/".

[44] G. Oikonomou and J. Mirkovic, "Modeling human behavior for defense against flash-crowd attacks," in *Proceedings of 2009 IEEE International Conference on Communications (ICC'09)*. IEEE, 2009, pp. 1–6.

[45] NSF, "Cloudlab," https://www.cloudlab.us, 2017.

[46] *Amazon Auto Scaling*. "https://aws.amazon.com/documentation/autoscaling".

[47] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage, "Inferring internet denial-of-service activity," *ACM Transactions on Computer Systems (TOCS)*, vol. 24, no. 2, pp. 115–139, 2006.

[48] *Amazon CloudWatch Concepts*. "http://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/cloudwatch_concepts.html".

[49] *Snort: The World's Most Widely Deployed IPS Technology*. "http://www.cisco.com/c/en/us/products/collateral/security/brief_c17-733286.html".

[50] *Snort.AD*. "http://www.anomalydetection.info/?,32".

[51] *Application DDoS Mitigation*. "Palo Alto Networks, Inc.", 2014.

[52] *Clavister DoS and DDos Protection.* "Clavister, Inc.", 2014.

[53] 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 11th international conference on World Wide Web.* ACM, 2002, pp. 293–304.

[54] *ASA Threat Detection Functionality and Configuration.* "http://www.cisco.com/c/en/us/support/docs/security/asa-5500-x-series-next-generation-firewalls/113685-asa-threat-detection.html".

[55] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic." in *Proceedings of Network and Distributed System Security Symposium (NDSS)*, 2008.

[56] Y. Xie and S.-Z. Yu, "Monitoring the application-layer ddos attacks for popular websites," *IEEE/ACM Transactions on Networking (TON)*, vol. 17, no. 1, pp. 15–25, 2009.

[57] S. Ranjan, R. Swaminathan, M. Uysal, A. Nucci, and E. Knightly, "Ddos-shield: Ddos-resilient scheduling to counter application layer attacks," *IEEE/ACM Transactions on Networking (TON)*, vol. 17, no. 1, pp. 26–39, 2009.

[58] R. Kohavi and R. Longbotham, "Online experiments: Lessons learned," *IEEE Computer Society*, 2007.

[59] H. Shan, Q. Wang, and Q. Yan, "Very short intermittent ddos attacks in an unsaturated system," in *Proceedings of the 13th International Conference on Security and Privacy in Communication Systems.* Springer, 2017.

[60] R. Rasti, M. Murthy, N. Weaver, and V. Paxson, "Temporal lensing and its application in pulsing denial-of-service attacks," in *Proceedings of 2015 IEEE Symposium on Security and Privacy (SP).* IEEE, 2015, pp. 187–198.

[61] E. Cambiaso, G. Papaleo, and M. Aiello, "Taxonomy of slow dos attacks to web applications," in *International Conference on Security in Computer Networks and Distributed Systems.* Springer, 2012, pp. 195–204.

[62] G. Maciá-Fernández, J. E. Díaz-Verdejo, P. García-Teodoro, and F. de Toro-Negro, "Lordas: A low-rate dos attack against application servers," in *Proceedings of International Workshop on Critical Information Infrastructures Security.* Springer, 2007, pp. 197–209.

[63] M. Bertoli, G. Casale, and G. Serazzri, "Java modelling tools: an open source suite for queueing network modelling andworkload analysis," in *Proceedings of the 3rd International Conference on Quantitative Evaluation of Systems (QEST'06).* IEEE, 2006, pp. 119–120.

[64] Amazon, "Amazon ec2," https://aws.amazon.com/ec2/, 2017.

[65] Microsoft, "Microsoft azure," https://azure.microsoft.com/en-us/?v=17.14, 2017.

[66] L. Kleinrock, *Queueing systems, volume 2: Computer applications*. John Wiley and Sons, New York, 1976, vol. 66.

[67] C.-A. Lai, J. Kimball, T. Zhu, Q. Wang, and C. Pu, "milliscope: a fine-grained monitoring framework for performance debugging of n-tier web services," in *Proceedings of the IEEE 37th International Conference on Distributed Computing Systems (ICDCS'17)*. IEEE, 2017, pp. 92–102.

[68] R. E. Kalman *et al.*, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[69] B. J. Odelson, M. R. Rajamani, and J. B. Rawlings, "A new autocovariance least-squares method for estimating noise covariances," *Automatica*, vol. 42, no. 2, pp. 303–308, 2006.

[70] S. Hiller, *Precise to the millisecond: NTP services in the "Internet of Things"*. "https://www.retarus.com/blog/en/precise-to-the-millisecond-ntp-services-in-the-internet-of-things/".

[71] D. Jayasinghe, S. Malkowski, Q. Wang, J. Li, P. Xiong, and C. Pu, "Variations in performance and scalability when migrating n-tier applications to different clouds," in *Proceedings of the IEEE International Conference on Cloud Computing (CLOUD'11)*. IEEE, 2011, pp. 73–80.

[72] C. Ye and K. Zheng, "Detection of application layer distributed denial of service," in *Proceedings of the IEEE International Conference on Computer Science and Network Technology*, vol. 1. IEEE, 2011, pp. 310–314.

[73] J. Yu, Z. Li, H. Chen, and X. Chen, "A detection and offense mechanism to defend against application layer ddos attacks," in *Proceedings of the IEEE 3rd International Conference on Networking and Services (ICNS'07)*. IEEE, June 2007, pp. 54–54.

[74] Y. Xuan, I. Shin, M. T. Thai, and T. Znati, "Detecting application denial-of-service attacks: A group-testing-based approach," *IEEE Transactions on parallel and distributed systems*, vol. 21, no. 8, pp. 1203–1216, 2010.

[75] *Amazon Elastic Load Balancing*. "https://aws.amazon.com/elasticloadbalancing/".

[76] *PhantomJS*. "http://phantomjs.org/".

[77] T. Peng, C. Leckie, and K. Ramamohanarao, "Survey of network-based defense mechanisms countering the dos and ddos problems," *ACM Computing Surveys (CSUR)*, 2007.

[78] R. R. Hansen, "Slowloris http dos," https://web.archive.org/web/20090822001255/http://ha.ckers.org/slowloris/, Accessed at July 9, 2016.

[79] S. A. O'Brien, *Widespread cyberattack takes down sites worldwide.* "http://money.cnn.com/2016/10/21/technology/ddos-attack-popular-sites/index.html".

[80] Z. Xu, H. Wang, and Z. Wu, "A measurement study on co-residence threat inside the cloud." in *Proceedings of the 24th USENIX Security Symposium*, 2015, pp. 929–944.

[81] T. S. Team, *Announcing Spotify Infrastructures Google Future.* "https://news.spotify.com/us/2016/02/23/announcing-spotify-infrastructures-googley-future/", 2017.

[82] H. Shan, Q. Wang, and C. Pu, "Tail attacks on web applications," in *Proceedings of the 24nd ACM SIGSAC Conference on Computer and Communications Security.* ACM, 2017.

[83] T. Vissers, T. Van Goethem, W. Joosen, and N. Nikiforakis, "Maneuvering around clouds: Bypassing cloud-based security providers," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security.* ACM, 2015, pp. 1530–1541.

[84] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes, "Cpi 2: Cpu performance isolation for shared compute clusters," in *Proceedings of the 8th ACM European Conference on Computer Systems.* ACM, 2013, pp. 379–391.

[85] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *Proceedings of the 44th annual IEEE/ACM International Symposium on Microarchitecture.* ACM, 2011, pp. 248–259.

[86] A. K. Maji, S. Mitra, and S. Bagchi, "Ice: An integrated configuration engine for interference mitigation in cloud services," in *Proceedings of 2015 IEEE International Conference on Autonomic Computing (ICAC).* IEEE, 2015, pp. 91–100.

[87] M. Kambadur, T. Moseley, R. Hank, and M. A. Kim, "Measuring interference between live datacenter applications," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis.* IEEE Computer Society Press, 2012, p. 51.

[88] H. Liu, "A new form of dos attack in a cloud and its avoidance mechanism," in *Proceedings of the 2010 ACM workshop on Cloud computing security workshop.* ACM, 2010, pp. 65–76.

[89] H. S. Bedi and S. Shiva, "Securing cloud infrastructure against co-resident dos attacks using game theoretic defense mechanisms," in *Proceedings of the International Conference on Advances in Computing, Communications and Informatics.* ACM, 2012, pp. 463–469.

[90] Q. Huang and P. P. Lee, "An experimental study of cascading performance interference in a virtualized environment," *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 4, pp. 43–52, 2013.

[91] F. Zhou, M. Goel, P. Desnoyers, and R. Sundaram, "Scheduler vulnerabilities and coordinated attacks in cloud computing," *Journal of Computer Security*, vol. 21, no. 4, pp. 533–559, 2013.

[92] V. Varadarajan, T. Kooburat, B. Farley, T. Ristenpart, and M. M. Swift, "Resource-freeing attacks: improve your cloud performance (at your neighbor's expense)," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 281–292.

[93] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: improving resource efficiency at scale," in *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3. ACM, 2015, pp. 450–462.

[94] *Amazon EC2 Instance Types*. "https://aws.amazon.com/ec2/instance-types/".

[95] *Microsoft Azure Virtual Machine Sizes for Cloud Services*. "https://docs.microsoft.com/en-us/azure/virtual-machines/windows/sizes-general".

[96] M. S. Inci, B. Gülmezoglu, G. I. Apecechea, T. Eisenbarth, and B. Sunar, "Seriously, get off my cloud! cross-vm rsa key recovery in a public cloud," *IACR Cryptology ePrint Archive*, vol. 2015, p. 898, 2015.

[97] *Microsoft Azure Virtual Machine for General purpose*. "https://docs.microsoft.com/en-us/azure/cloud-services/cloud-services-sizes-specs".

[98] A. Herdrich, E. Verplanke, P. Autee, R. Illikkal, C. Gianos, R. Singhal, and R. Iyer, "Cache qos: From concept to reality in the intel® xeon® processor e5-2600 v3 product family," in *Proceedings of 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2016, pp. 657–668.

[99] V. Varadarajan, Y. Zhang, T. Ristenpart, and M. M. Swift, "A placement vulnerability study in multi-tenant public clouds," in *Proceedings of the 24th USENIX Security Symposium*, 2015, pp. 913–928.

[100] L. Litty, H. A. Lagar-Cavilla, and D. Lie, "Computer meteorology: Monitoring compute clouds." in *HotOS*, 2009.

[101] *Kernel Virtual Machine*. "https://www.linux-kvm.org/page/Main_Page".

[102] *RAMspeed, a cache and memory benchmarking tool*. "http://alasir.com/software/ramspeed".

[103] S. Adler, "The slashdot effect: an analysis of three internet publications," *Linux Gazette*, 1999.

[104] L. Kleinrock, *Queueing Systems*. Wiley, 1975.

[105] D. G. Kendall, "Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded markov chain," *The Annals of Mathematical Statistics*, pp. 338–354, 1953.

[106] Apache, *Apache MPM worker.* "https://httpd.apache.org/docs/2.4/mod/worker.html", 2017.

[107] A. O. Allen, *Probability, statistics, and queueing theory.* Academic Press, 2014.

[108] M. Harchol-Balter, *Performance modeling and design of computer systems: queueing theory in action.* Cambridge University Press, 2013.

[109] S. Votke, S. A. Javadi, and A. Gandhi, "Modeling and analysis of performance under interference in the cloud," in *Proceedings of the 25th IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS).* IEEE, 2017.

[110] Amazone, *AWS Elastic Load Balancing.* "https://aws.amazon.com/elasticloadbalancing/", 2017.

[111] L. man page, *xentrace(8).* "https://linux.die.net/man/8/xentrace", 2017.

[112] "Oprofile," "http://oprofile.sourceforge.net/".

[113] T. Moscibroda and O. Mutlu, "Memory performance attacks: Denial of memory service in multi-core systems," in *Proceedings of 16th USENIX Security Symposium.* USENIX Association, 2007, p. 18.

[114] M. S. Inci, T. Eisenbarth, and B. Sunar, "Hit by the bus: Qos degradation attack on android," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security.* ACM, 2017, pp. 716–727.

[115] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," in *ACM SIGPLAN Notices*, vol. 48, no. 4. ACM, 2013, pp. 77–88.

[116] D. Novakovic, N. Vasic, S. Novakovic, D. Kostic, and R. Bianchini, "Deepdive: Transparently identifying and managing performance interference in virtualized environments," in *Proceedings of the 2013 USENIX Annual Technical Conference*, no. EPFL-CONF-185984, 2013.

[117] D. Sites, *Data Center Computers: Modern challenges in CPU design.* "https://www.cs.wisc.edu/events/1887", 2017.

APPENDIX A
PERMISSION TO PUBLISH CHAPTER 2

EAI Copyright and Consent Form

Title of Work: Very Short Intermittent DDoS Attacks in an Unsaturated System

Complete list of authors: Shan, Huasong<hshan1@lsu.edu> (Louisiana State University), Wang, Qingyang<qwang26@lsu.edu> (Louisiana State University), Yan, Qiben<qyan@cse.unl.edu> (University of Nebraska-Lincoln)

EAI publication title: Proceedings of 13th EAI International Conference on Security and Privacy in Communication Networks

Transfer of Copyright Agreement

Copyright to the Work, to any supplemental material integral to the Work which is submitted with it for review and publication such as an extended proof or supplementary text and figures, and to any subsequent errata, is hereby according to the rules set in the U.S. Copyright Act, 17 U.S.C., transferred to EAI for the full term throughout the world, subject to the Author Rights (as hereinafter defined) and to the acceptance of the Work for publication by EAI. This transfer of copyright includes all material to be published as part of the Work, including but not limited to tables, figures, graphs, movies, other multimedia files, and all supplemental materials. EAI shall have the right to register copyright in the Work in its name as claimant, whether separately or as part of the journal issue, book, volume or other medium in which the Work is 101 included.

The author(s), and in the case of a Work Made For Hire, as defined in the U.S. Copyright Act, 17 U.S.C. §101, the employer named below, shall have the following rights (the "Author Rights"):

1. All proprietary rights other than copyright, such as patent.
2. The right to reuse any portion of the Work, without fee, in future works of the author(s) or employer, including books, lectures and presentations in all media, provided that a citation of the EAI -published work, notice of the Copyright, and EAI DOI are included.
3. The right to make, and hold copyright in, works derived from the Work, as long as the following conditions are met: (i) at least one author of the derivative work is an author of the Work; (ii) the derived work includes at least 30% of new material not covered by EAI's copyright in the Work. If these conditions are met, copyright in the derivative work rests with the authors of that work, and EAI and its successors and assigns will make no claim on that copyright. If these conditions are not met, explicit EAI permission must be obtained. Nothing in this Section shall prevent EAI and its successors and assigns from exercising its rights in the Work.
4. The right to post and update author-prepared versions of the article on free-access e-print servers, including the author and/or employer's home page and any repository legally mandated by the agency funding the research on which the Work is based. If the author wishes the EAI-prepared version to be used for an online posting, permission is required from EAI; if granted, use will be subject to EAI terms and conditions.

General Terms

- The undersigned represents that he/she has the power and authority to make and execute this assignment;
- The undersigned agrees to indemnify and hold harmless EAI from any damage or expense that may arise in the event of a breach of any of the warranties set forth above.
- In the event that the Work is not accepted and published by EAI or is withdrawn by the author(s) before acceptance by EAI, the foregoing copyright transfer shall become null and void.
- For jointly authored works, all joint authors should sign, or one of the authors should sign as an authorized agent for the others. If the article has been prepared as a Work Made for Hire, as defined in the U.S. Copyright Act, 17 U.S.C. §101, the transfer should be signed by the employer.

Signature(s)

A. Single Author/Authorized Agent for Joint Authors

To sign the agreement, please enter your name, job title and employer in the box provided.

signature ***Huasong Shan*** 2017/09/28

Date

B. Employer authorized signature (for Work Made for Hire)

signature 2017/09/28
Date

C. U.S. Government Employee Certification (where applicable)

This will certify that all authors of the Work are U.S. government employees and prepared the Work on a subject within the scope of their official duties. As such, the Work is not subject to U.S. copyright protection. (Authors should still sign signature line [A] above to enable EAI to claim and protect its copyright in international jurisdictions.)

signature 2017/09/28
Date

D. Crown Copyright Certification (where applicable)

This will certify that all authors of the Work are employees of the British or British Commonwealth Government and prepared the Work in connection with their official duties. As such, the Work is subject to Crown Copyright and is not assigned to EAI as set forth in the first sentence of the Transfer of Copyright Agreement above. The undersigned acknowledges, however, that EAI has the right to publish, distribute and reprint the Work in all forms and media. (Authors should still sign signature line [A] above to indicate their acceptance of all terms other than the copyright transfer.)

signature 2017/09/28
Date

☒ **I have read and agree to the forms of the EAI Copyright and Consent Form.**

APPENDIX B
PERMISSION TO PUBLISH CHAPTER 3

RE: Permission Request_author reuse [ref:_00D30oeGz._5000c1Q7Os6:ref]

noreply@salesforce.com on behalf of customercare@copyright.com

Mon 10/2/2017 9:30 AM

To: Huasong Shan <hshan1@lsu.edu>;

October 2, 2017

Dear Huasong Shan:

Thank you for your email to Rightslink. I've contacted the publisher about your situation and ACM has let me know that according to ACM Publishing Policy, the author doesn't need permission to include his own paper in his thesis/dissertation. See Sec. 2.5 at www.acm.org/publications/policies/copyright-policy#permanent for details.

However if your institution requires it, you must wait until the paper is published on Oct. 30 to obtain a free license that allows for his and his university library's use of the paper as part of his dissertation.

The papers of CCS '17 will automatically be made available through RightsLink once published.

Thank you for your patience.

Sincerely,
Christine
Christine M. Zoro
Customer Account Specialist
Copyright Clearance Center
222 Rosewood Drive
Danvers, MA 01923
+1.855.239.3415 Customer Service Toll Free

---------- Original Message ----------
From: Huasong Shan [hshan1@lsu.edu]
Sent: 9/28/2017 10:51 AM
To: customercare@copyright.com
Subject: Permission Request

To whom it may concern,

I am a first author of the paper published in ACM CCS 2017 with the title of "Tail Attacks on Web Applications" with DOI: https://doi.org/10.1145/3133956.3133968.
I would like to include some parts of this paper in one of the chapters in my Ph.D. dissertation.
I am wondering how to receive such a permission from ACM.

Thanks in advance.

Kindly Regards,

Huasong Shan | Ph.D. Student
Computer Science and Engineering Division
Louisiana State University
Baton Rouge, LA 70803

ref:_00D30oeGz._5000c1Q7Os6:ref

VITA

Huasong Shan was born in 1981, in Hubei, China. He received his Bachelor and Master of Science degree in computer science from Huazhong University of Science and Technology, Hubei, China in 2003 and 2006, respectively. He then worked as a software test engineer at Zhongxing Telecommunication Equipment Corporation, Shanghai, China for half and a year. From 2008 to 2015, he worked as a software engineer at Spreadtrum Communications Incorporated Company, Shanghai, China on the infrastructure team of protocol software division. In August 2015 he came to Louisiana State University, Baton Rouge, Louisiana to start graduate studies in computer science. He is a candidate for the degree of Doctor of Philosophy in computer science in fall 2017.