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

A Dissertation Proposal

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

School of Electrical Engineering and Computer Science

by
Huasong Shan
July 2017

Table of Contents

List of Tables

List of Figures

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 can not 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 thesis proposal, 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 network queuing theory and implementing the attacking framework based-on feed-back control theory. We propose the extended work of internal attacks, and investigate the feasibility of 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]. On October 21, 2016, A number of popular websites like Twitter and Netflix shut down for some users worldwide, due to two DDoS attacks on the DNS servers of Dyn company [3]. 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" [4] 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 [5–7]. 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 [8,9] 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 [10,11]; and (ii) the dependence among various resources is very strong, even in different physical machines [12]; and (iii) resource contention occurs when the utilization becomes high [13, 14]; and (iv) the source of bottleneck is usually dynamic and unpredictable [15]; and (v) the adverse impact of application performance is very remarkable [16–18]. These new identified performance problems in the cloud motivated me to study the practicality and impact of the hypothetical attacks, especially for Web applications.

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% [19]; Google requires 99 percentage of its queries to finish within 500ms [20]. In practice, the tail latency, rather than the average latency, is of particular concern for mission-critical web-facing applications [20–23]. 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 [21,24,25]. In this proposal, 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 [8,9,26]. In the cloud environments, the attacking approaches are much more flexible and abundant, can be categorized in two classes: external and internal attacks [27,28]. External attacks are the most orthodox form of DoS [8, 9, 26]. For external attacks, our hypothesis is that the external HTTP requests supported by the victim web-sites 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 [27, 28]. For internal attacks, our hypothesis is that the adversarial programs in the co-located VMs [29] (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 [30, 31].

In this proposal 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.

Before proceeding to the concrete contributions of my thesis, my thesis statement can be formulated as follows:

Thesis Statement: *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 my thesis statement, we make the following three contributions:

- We describe external and internal attacks in a stealthy manner that can broadly threaten a wide range of web applications deployed in the cloud. As for external attacks, unlike the traditional brute-force DoS attacks or pulsating attacks which focus on network bandwidth, the external attacks use legitimate HTTP requests to attack the bottleneck resource of the node in the cloud hosting the target web system, thereby reducing the cost of an effective attack while keeping the attack highly stealthy. This attack 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). Since the average utilization of the target system under attack is usually in moderate level (e.g., about 50-60%) and all the attacking requests are legitimate, tracing the cause of the performance problem is difficult.

  As a second step, we plan to implement the internal attacks in a stealthy manner 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 also 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. We expect the proposed internal attacks can like the implemented external attacks, the average utilization of the target system under attack is far from saturation, and all the behaviors in the attacking VMs are legitimate, tracing the source of the performance problem caused by the internal attacks is more difficult than the external attacks.

- We develop the attacking framework that is able to efficiently and adaptively launch the proposed attacks against a target web application. For the external attacks, we formulate the impact of our attacks in n-tier systems based-on queue network models, which can effectively guide the external attacks in an even stealthy way. And then we implement the attacking framework based-on the feed-back 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 [32]) under realistic cloud scaling settings and equipped with the most popular state-of-the-art DDoS defense tools, we validate the practicality of the external attacking framework. 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, even though the web application benchmark is deployed in the elastic public cloud platform (e.g., Amazon EC2, Microsoft Azure).

  As a second step, we propose to implement the feed-back control framework for the internal attacks to efficiently and stealthily trigger the internal resource contention in the cloud by dynamically controlling the optimal attack parameters of the attacking VMs. We expect the control framework for the internal attacks also can achieve the same damage and stealthiness as the one for the external attacks.

- 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.

  As a second step, 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). The challenging problem is the tradeoff between the detect accuracy and the overhead incurred by the monitoring tools.

We outline the remainder of this proposal as follows. Chapter 2 describes the completed work, the external attacks outside the cloud. I will highlight the unique aspects of the proposed attacks, showing the experimental results with the benchmark web applications RUBBoS [32]. Chapter 3 shows some preliminary experimental results for the internal attacks, and proposes my agenda of remaining dissertation research work. Chapter 4 concludes the proposal and discuss some future work.

Chapter 2
Completed Work: External Attacks

This chapter presents two parts of my core thesis research that have already been completed. First, Section 2.1 illustrates a new low-volume application-layer external attack that can broadly threaten a wide range of web applications in a stealthy manner [33]. Section 2.2 formulates the impact of external attacks in n-tier systems based-on queue network models, and implements a feed-back control-theoretic (e.g., Kalman filter) attack framework that can dynamically tuning the optimal attack parameters to fit the dynamics of background requests or system state [34].

2.1 External Attacks: Very Short Intermittent External DDoS Attacks

2.1.1 Attack Origin

Very Short Intermittent DDoS (VSI-DDoS) Attacks originate from the new phenomenon of very short bottlenecks (VSBs), also called transient bottlenecks in recent performance studies of web applications deployed in cloud [10, 11]. 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

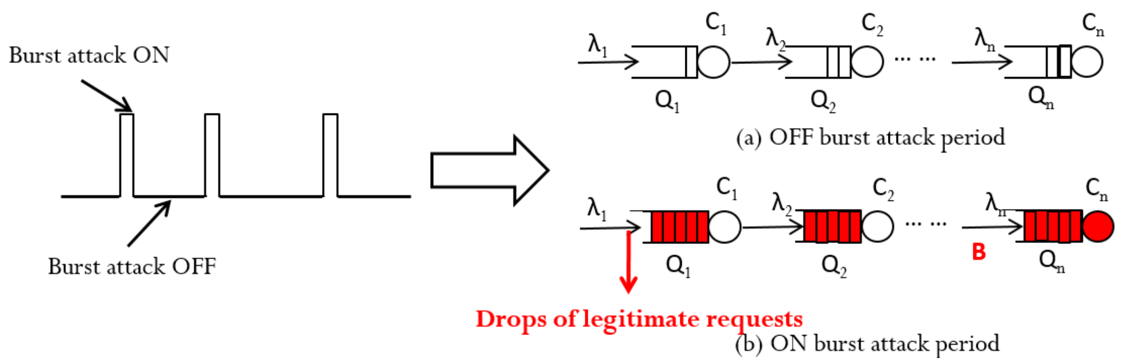


FIGURE 2.1: **Attack scenario and system model**

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, causing TCP retransmissions (minimum time-out is 1 second [35]). 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 sudden jump of resource demand flowing into the target system and cause VSBs in the weakest point of the 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.

2.1.2 Attack Scenario

Consider a scenario of a VSI-DDoS attack in an n-tier system in Figure 2.1. The detailed model analysis and experimental data of VSI-DDoS attacks are in the following sections. By alternating short "ON" and long "OFF" attack burst, the attackers guarantee the attacks 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 VSI-DDoS 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 shown in Figure 2.1b. (**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 [35]), the end users with the requests encountered 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 2.1a. Unlike the traditional flooding DDoS attacks which aim to bring down the system, our attack aims to degrade the

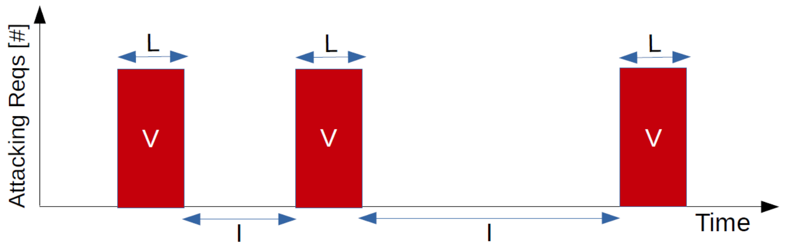


FIGURE 2.2: **An illustration of a VSI-DDoS attack**, which consists of burst volume V of HTTP requests, burst length L, and burst interval I.

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 tradeoff between attack damage and elusiveness.

2.1.3 Attack definition

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 [36–38] or decentralized methods [39], to coordinate bots to send synchronized traffic and cause network congestion at a specific link. Our focus in this paper 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 2.2):

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

where,

- $Effect$ is the measure of attacking effectiveness; we use percentile response time as a metric to measure the tail latency of the target 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 [8,9].

- 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.

2.1.4 Experimental Results

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.

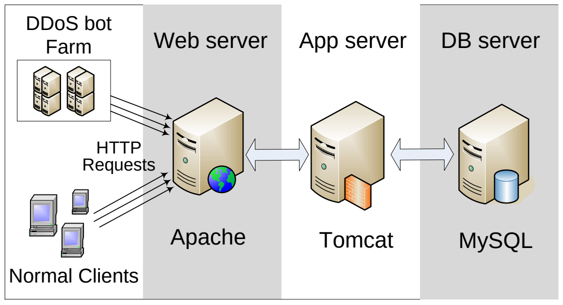
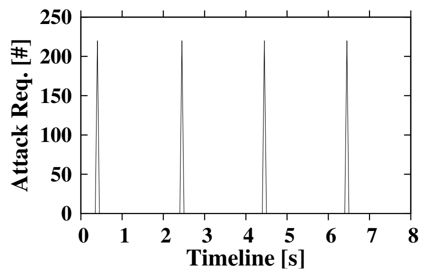


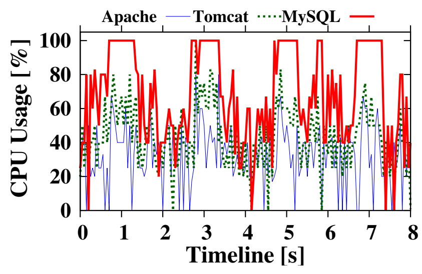
FIGURE 2.3: **Experimental sample topology**

Benchmark Measurements. We adopt RUBBoS [32], a representative n-tier web application benchmark modeled after the popular news website *Slashdot*. In Figure 2.3, 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. 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 webpages. Meanwhile, we use *Apache Bench* to send intermittent bursts of carefully chosen legitimate HTTP requests; each burst are sent out within a very short time period (e.g., 50ms).

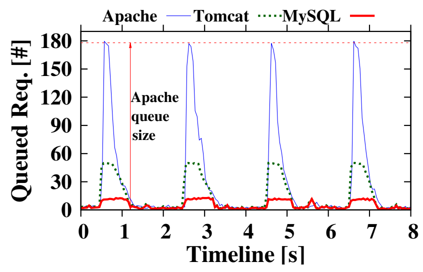
The mechanism of how VSI-DDoS attacks impact the performance of the target n-tier web system can only be seen using fine-grained monitoring. Figure 2.4 shows such an analysis when the target 3-tier benchmark website serving 3000 legitimate users is under a VSI-DDoS attack. All the metrics in the subfigures are measured at every 50ms time window. Figure 2.4a 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 2.4b. These



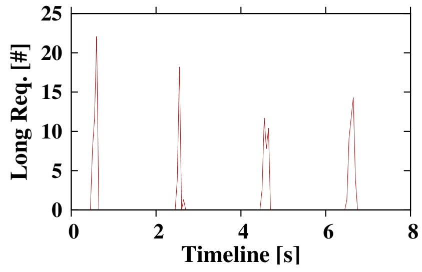
(a) Bursts of attacking HTTP requests in a VSI-DDoS attack during an 8-second time period. We count the # of attacking requests in every 50ms time window.



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



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

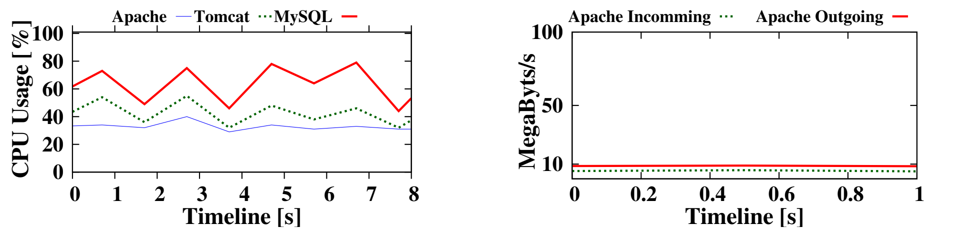


(d) Number of requests (from legitimate users) with response time > 1s during the same 8-second time period. Such long requests are caused by TCP retransmissions once the front-most Apache drops incoming requests (see (c)).

FIGURE 2.4: **Measured performance of the benchmark application under a VSI-DDoS attack.** Bursts of attacking HTTP requests (a) trigger VSBs in the bottom-most MySQL of the system (b), which cause requests to queue from local to the front-most Apache (c). Queue-overflows occur in Apache, causing TCP retransmissions and long response time requests (d).

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 2.4c. We call this phenomenon as *push-back wave*. 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 2.4d.

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.



(a) CPU util. of each server during the same 8s attacking period as in Fig. 2.4.

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

FIGURE 2.5: **Resource utilization of each server in the system under the VSI-DDoS Attack.** Metrics are measured using *vmstat* at every 1 second. (a) and (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 2.5a 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 2.5b 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.

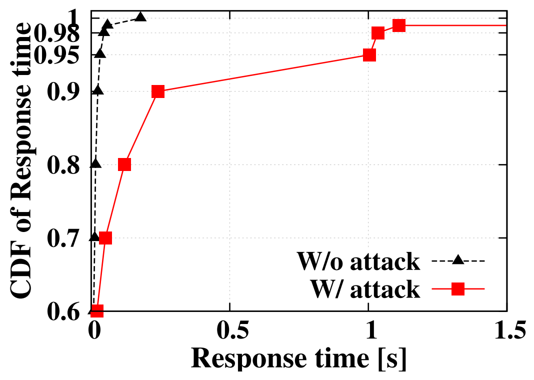


FIGURE 2.6: **Percentile response time of the system serving 3000 legitimate users without and with the attack.** The 95th percentile response time under attack is $>$ 1 second, clearly showing the long-tail latency problem caused by the VSI-DDoS attack.

To illustrate the impact of the VSI-DDoS attack, we compare the tail latency of the target system under attack and without-attack in Figure 2.6. The percentile response time of the system under attack (the red line) uses the same dataset in Figure 2.4. 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 considered as severe performance degradation by most modern e-commerce web applications (e.g., Amazon).

2.2 External Attacks: Modeling and Control Framework

2.2.1 External Attack 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.

Queue network models are commonly used to analyze the performance problems in complex computer systems [40], 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 VSI-DDoS attacks shown in Figure 2.1. Table 2.1 summarizes the notation and description of the parameters used in our model. The basic attack pattern (see Event1 in Section 2.1.2) shown in Figure 2.1 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 VSI-DDoS attack. If all the attack requests will not be dropped by the target system , we can calculate the attack volume during a burst by:

Param.	**Description**
Q_i	the queue size for the ith tier
$C_{i,A}$	the capacity serving attack request for the ith tier
$C_{i,L}$	the capacity serving legitimate request 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

TABLE 2.1: **Model parameters**

$$V = B * L \tag{2.2}$$

We assume that the external burst of legitimate HTTP requests (Event1 in Section 2.1.2) can cause sudden jump of resource demand flowing into the target system and cause millibottlenecks (Event2 in Section 2.1.2) in the weakest point of the system [36]. 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 2.1.2) caused by overloaded attack bursts can lead to cross tier queue overflow from downstream tiers to upstream tiers (Event3 in Section 2.1.2) 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

$$l_n = \frac{Q_n}{(\lambda_n + B - C_{n,A})} \tag{2.3}$$

$$l_{n-1} = \frac{(Q_{n-1} - Q_n)}{(\lambda_{n-1} + \lambda_n + B - C_{n,A})} \tag{2.4}$$

$$...$$

$$l_1 = \frac{(Q_1 - Q_2)}{(\sum_{i=1}^{n} \lambda_i + B - C_{n,A})} \tag{2.5}$$

When millibottlenecks occur in the n-th server, firstly the queue in the n-th tier is overflown during l_n, and then the next queue in the *n-1*-th tier is overflown during l_{n-1}, and so on. Thus, in order to overflow all the queues in the n-tier system, the required time is the sum of l_i. Here, Q, C, and λ are constants. l_i is a function of *B*.

Once all the queues are overflown (Event3 in Section 2.1.2) in the n-tier system, the new incoming requests may be dropped by the front tier (Event4 in Section 2.1.2). 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, then we can approximately infer *damage length* by:

$$P_D = L - \sum_{i=1}^{n} l_i \tag{2.6}$$

Further, the end-users with the dropped requests perceive very long response time (Event5 in Section 2.1.2), leading to the long-tail latency problem caused by our attacks (Event1 in Section 2.1.2), which can be approximately estimated as follows,

$$\rho(\mathrm{T}) = \frac{P_D}{T} \tag{2.7}$$

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. Thus, *millibottleneck length* should involve the resource consumption for both attack and normal requests during a burst. Equation (2.8) represents *millibottleneck length* derived through the geometric progression in mathematics.

$$\begin{aligned} P_{MB} &= V * \frac{1}{C_{n,A}} + V * \frac{1}{C_{n,A}} * \lambda_n * \frac{1}{C_{n,L}} \\ &\quad + V * \frac{1}{C_{n,A}} * \lambda_n * \frac{1}{C_{n,L}} * \lambda_n * \frac{1}{C_{n,L}} + \ldots \\ &\quad + V * \frac{1}{C_{n,A}} * (\lambda_n * \frac{1}{C_{n,L}})^m \\ &= \lim_{m\to\infty} \sum_{k=0}^{m} V * \frac{1}{C_{n,A}} * (\lambda_n * \frac{1}{C_{n,L}})^k \\ &= V * \frac{1}{C_{n,A}} * \lim_{m\to\infty} \frac{(1 - (\lambda_n * \frac{1}{C_{n,L}})^{m+1})}{(1 - (\lambda_n * \frac{1}{C_{n,L}}))} \\ &= V * \frac{1}{C_{n,A}} * \frac{1}{(1 - (\lambda_n * \frac{1}{C_{n,L}}))} \end{aligned} \tag{2.8}$$

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

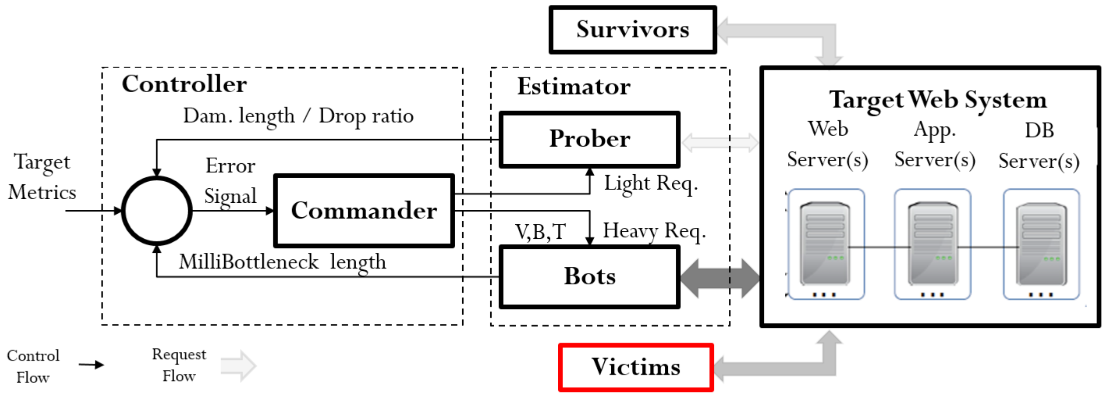


FIGURE 2.7: **A feedback control framework**

2.2.2 External Attack Framework

As mentioned before, the analytical model 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 requests can be dropped, etc.). However, our attacks should consider more realistic case with dynamics of baseline workload [1] 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 VSI-DDoS 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 VSI-DDoS 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. We implement a feed-back control-theoretic (e.g., Kalman filter) framework that allows attackers to fit the dynamics of background requests

[1]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 by dynamically adjusting the optimal attack parameters in Figure 2.7.

2.2.3 External Attack Mitigation

Here, we consider a solution to detect and defend against VSI-DDoS 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 2.1.2). We present a workflow to mitigate our attacks involving three stages: fine-grained monitoring, burst detection, and bots blocking.

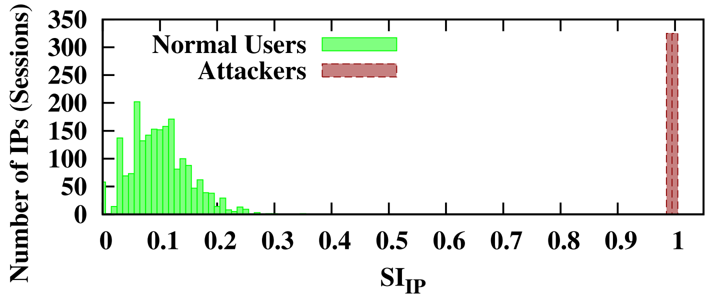


FIGURE 2.8: Identify bots

Fine-grained monitoring. 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 [10–12]. 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 requirement of eliminating the special millibottlenecks.

Burst detection. 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 2.1.2, 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.

IP-based statistical analysis defense. Once we identify the bursts of VSI-DDoS 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 2.1.2, we can ideally define a new request metric that quantifies the level of suspicion index 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:

$$\mathrm{SI}_{IP} = \frac{\mathrm{NB}_{IP}}{\mathrm{N}_{IP}} \tag{2.9}$$

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 2.8 shows Probability Density Function of SI_{IP} in the RUBBoS experiment in Section 2.1.4. The red bar represents the bots and the green bars represent the 2000 legitimate users. In this way to identify bots, the false positive and false negative error can be close to 0 with 100% high precision.

Chapter 3
Proposed Work: Internal Attacks

The remaining and ongoing work of this dissertation research will extend our research on *Very Short Intermittent DDoS Attacks* launched from external HTTP requests outside the cloud to internal resource contention between the co-located Virtual Machines inside the cloud. Our preliminary experimental results show that the response time of the target Web application can be heavily influenced by its co-located attacking applications inside the cloud in Section 3.3. Base-on the completed work, we plan to do the ongoing research in the timeline shown in Table 3.4.

3.1 Background

IaaS Instance Types. Infrastructure-as-a-Service (IaaS) cloud computing providers, such as Amazon EC2 [41] and Microsoft Azure [42], 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 [13, 14] for the users' payment, even the leakage of the private information [29, 43]. The fundamental reason is the ubiquitous phenomenon of performance interference between the co-located instances (in the same host) inside the cloud [15, 44, 45].

Performance Interference. 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 [15, 31, 45]. 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 [41] and Microsoft Azure [46] (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 this type of physical processors, Last Level Cache (LLC) [47] 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 [44].

2. *Resource overbooking:* To maximum the revenue for the cloud providers, except for sharing hardware resources, they sometimes overbook their servers for their tenants. [24] claims a vCPU for Amazon EC2 small instances only can use at most 40% of a physical core due to overbooking the physical cores.

Solutions to mitigate interference. Existing solutions try to detect and mitigate the interference from two levels based on the low-level hardware counts (e.g., CPI): infrastructure reconfiguration [44, 48] and application reconfiguration [13, 15, 45]. In the paper [48], the default sampling duration is 10 seconds and sampling frequency is every 1 minute, in this configuration they can detect the interference

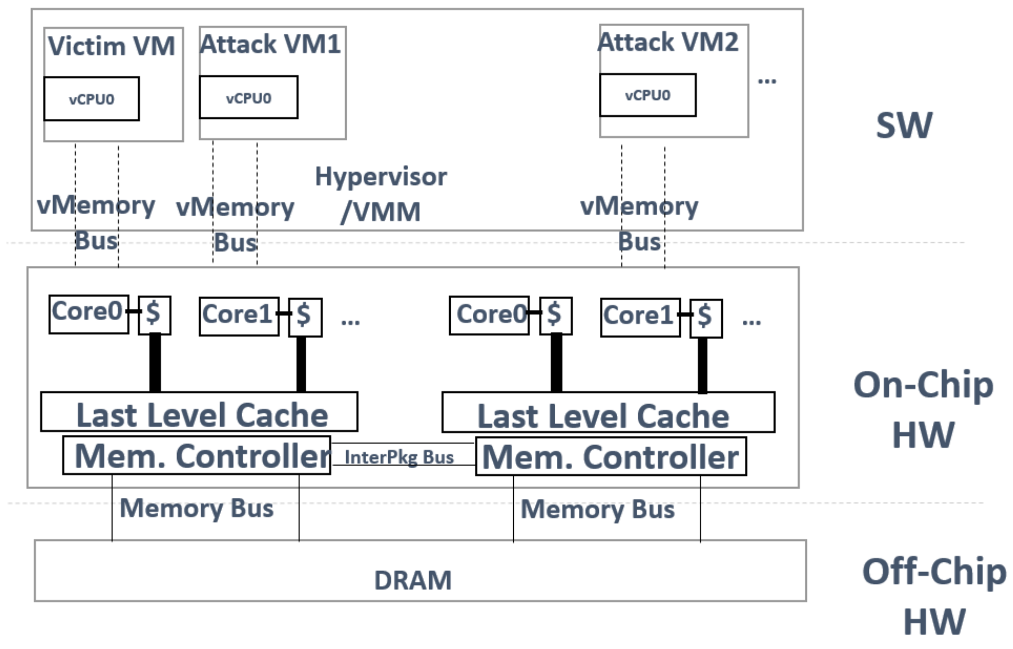


FIGURE 3.1: **Shared On-Chip Memory Resource Contention.**

on the order of several minutes. Maji et al. [15, 45] can deal with the short-lived cases that saturate server resources lasting for less than a minute, the sampling interval is 5 second. Because of the high overhead of tracing the low-level hardware counts [48], the tracing granularity can not be fine-grained; on the other hand, a small sampling interval incurs higher noises [45].

Shared On-Chip Memory Resource Contention. 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. The commercial cloud providers can assign the Memory size and CPU cores to specific instance for VM [41, 42], 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 [49]; 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 my thesis proposal, 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 (we term it *MemCA* attacks, as one type of *internal VSI-DDoS attacks)*, and leave exploration of other types of resource contentions to future work.

3.2 Goals 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 [7, 29, 50], 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, as is the case for today's public IaaS cloud.

3.3 Preliminary Experimental Results

3.3.1 Memory Bandwidth Contention Measurements

Experiment Methodology. 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 [41]. This private cloud is managed by Openstack Ocata; each VM, running CentOS release 6.7, is managed by the hypervisor KVM [51].

The adversarial program to saturate memory bandwidth of the host, is RAMspeed, a cache and memory benchmarking tool [52]. RAMspeed supports multi processes to give more stress 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, [24] 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

TABLE 3.1: **Measured Used Memory Bandwidth with multi VMs, RAM-speed specifies one thread to run.**

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

MemBW: Memory Bandwidth (MB/s)

Memory Bandwidth Contention among VMs. When RAMspeed runs with one thread, it can saturate the CPU usage. Table 3.1 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.

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 3.2 shows memory bandwidth of one VM with 2 vCPUs measured by RAMspeed with multi-thread cases. When threads

TABLE 3.2: **Measured Used Memory Bandwidth in a VM with 2 vCPUs, RAMspeed runs with multi threads.**

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

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.

Takeaway. Through the previous experiments, we profile the capacity of servers hosting VMs and pressure level caused by the adversarial. To trigger an effective contention of memory bus, 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 the same core, next is in the same package, last is in the different package. This implication is that memory bus can be split into three sections, including in one core, in same package, in all packages, as shown in Figure 3.1. The capacity for each one increases from top to bottom, as shown in Figure 3.1.

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 [44, 48] and short-lived [15, 45] interference detection, our goal is to create very-short-lived inference burst.

3.3.2 MemCA attacks on Local Testbed

Experiment Methodology. We adopt RUBBoS [32], a representative n-tier web application benchmark modeled after the popular news website *Slashdot* [53]. We use a typical 3-tier configuration, with 1 Apache web server, 1 Tomcat application Server, and 1 MySQL database server. Figure 3.2 shows RUBBoS sample topology in our controlled private cloud. RUBBoS has a workload generator to emulate

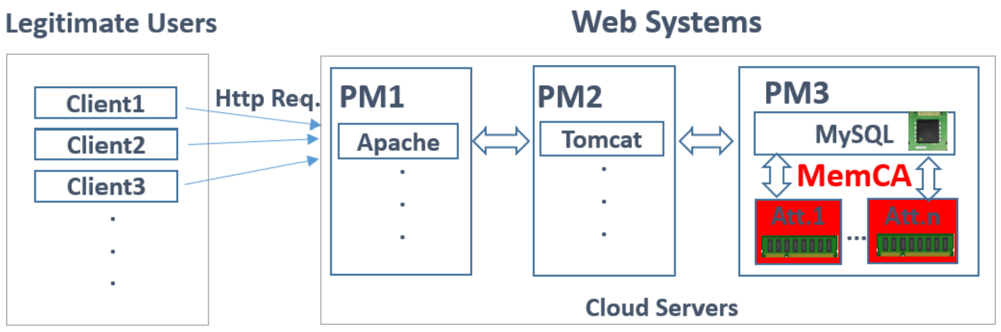


FIGURE 3.2: **Experimental sample topology**

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 widely investigated [24], the different package case can easily extended only by increases more VMs or vCPUs. Our attacking goal is to cause the SLO violation (99ile response time over hundreds of milli-seconds), we gradually increase the number of co-located attacking VMs until the attacking goal is achieved.

Results. Table 3.3 lists the corresponding response time and the CPU utilization of MySQL under different workloads and attacking intensity. Intuitively and ideally, the performance of the target VM should be the same even in the highly interference co-located case if the isolation in the cloud is enough perfect. However, it is far from the case. When we run RAMspeed in the adversarial VM, to

TABLE 3.3: **Observations of Response time and Potential Bottleneck resource (MySQL CPU utilization) in Controlled 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

achieve the SLO of the latency-intensive workloads (99ile response time within tens of milli-seconds [54]), the CPU utilization of the target VM MySQL increases as the number of attacking 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 lower workload requires fewer working set of memory and cache, and the threshold of triggering memory contention is much higher, leading to the requirement of more attacking VMs due to high probability of consuming more memory bus with more attacking VMs.

Analysis of the Contention caused by MemCA. Table 3.3 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 [13] triggered by our adversarial program, leading to more CPU time to service the cache misses.

3.4 Proposed Internal Attack Framework

Through the experiments in Section 3.3, 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 [15, 45, 48], 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 3.3):

$$Effect = \mathbb{A}(V, L, I) \tag{3.1}$$

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

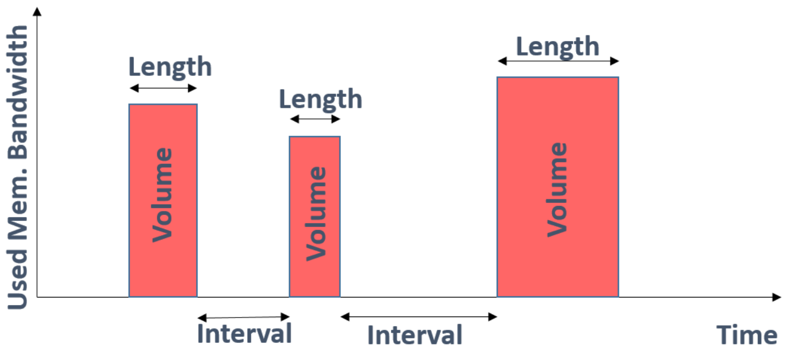


FIGURE 3.3: **An illustration of a MemCA attack**, which consists of interference burst volume V, length L, and interval I.

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 lasting period (length) of each interference burst. L should be short enough to guarantee the interference burst not to be captured by the interference detection mechanisms [15, 45, 48].
- 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.

To effectively launch and control our attacks, we propose an attack framework including two parts: MemCA front end (MemCA-FE) and MemCA back end (MemCA-BE)(Figure 3.4). MemCA-FE creates co-located VMs with the target

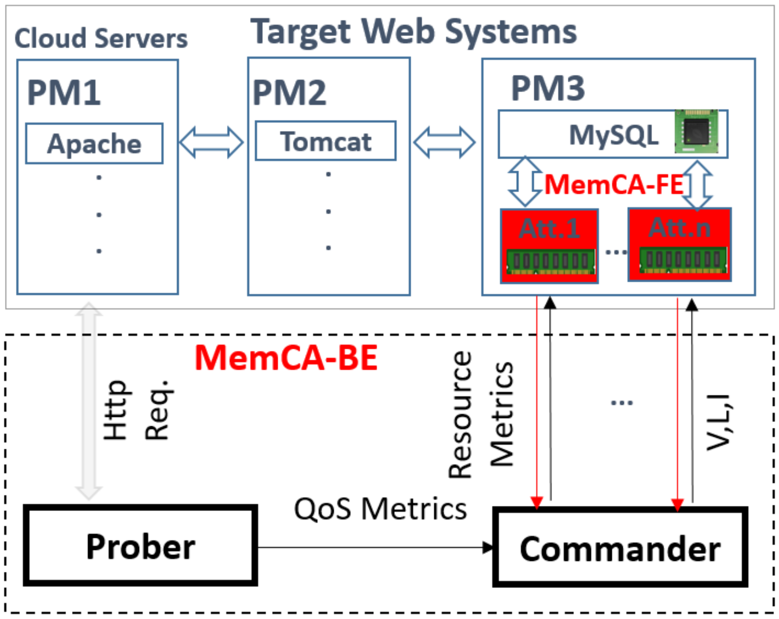


FIGURE 3.4: **Proposed Internal Attack Framework.**

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.

3.5 Proposed Work and Schedule

TABLE 3.4: **Proposed research schedule**

Research Activity	Timeline
Implement a control framework to effectively mount Internal Very Short Intermittent DDoS Attacks	Jul.-Sep. 2017
Provide a solution for mitigating the proposed Internal Very Short Intermittent DDoS Attacks	Oct. 2017
Write and defend my dissertation	Nov.-Dec. 2017

The remaining and ongoing work of this dissertation research will extend our research on very short intermittent DDoS Attacks on availability of Web applications launched from the co-located VMs inside the cloud. We plan to do the research in the timeline shown in Table 3.4. Specifically, the proposed work includes two aspects:

- **Implement a control framework to effectively mount Internal Very Short Intermittent DDoS Attacks.**

  We plan to implement the internal attacks in a stealthy manner 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. The goal of this internal attack is to create 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. Our preliminary experimental results show that in a co-located scenario, the response time of the target Web application can be heavily influenced by its collocated attacking applications. We expect the proposed internal attacks can like the implemented external attacks, the average utilization of the target system under attack is also far from saturation, which can be even more stealthy than the external attacks due to no trace of external traffic to the target system. We also expect the proposed attack framework for the internal attacks to efficiently and stealthily trigger the internal resource contention in the cloud by dynamically controlling the optimal attack parameters of the attacking VMs.

- **Provide a solution for mitigating the proposed Internal Very Short Intermittent DDoS Attacks.** Due to the unique feature of the proposed attacks (very short-lived and intermittent), we should exploit fine-grained monitoring tools to record the performance count in the host to capture the interference. The potential approach is to record the performance count in the host, identify the source of actual performance interference, and reschedule the placement of the adversarial and victim VMs (e.g., live migration). The challenging problem is the tradeoff between the detect accuracy and the overhead incurred by the monitoring tools.

Chapter 4
Conclusion and Future Work

4.1 Conclusion

Despite of the benefits of cloud computing, many research efforts find the system vulnerability in the cloud in Section 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.

Through the experimental results verifying the damage of a series of the proposed attacks (external attacks and internal attacks), I prove the feasibility that this work can efficiently and stealthily degrade the performance of the target Web applications deployed in the cloud. I have not only completed the research work on external attacks (see Section 2), my preliminary experimental results also show that the response time of the target Web application can be heavily influenced by its co-located attacking applications inside the cloud. Thus, the proposed work and plan (see Section 3) should be feasible and reasonable.

Overall, through highlighting the practicality and impact of the proposed 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.

4.1.1 Peer-Reviewed Papers

A list of peer-reviewed papers listed by completed works is below.

External DDoS Attacks on Availability of Web Services in the Cloud

- Shan, H., Wang, Q., and Yan, Q., "Very Short Intermittent DDoS Attacks in an Unsaturated System," submitted to SECURECOMM '17.

- Shan, H. and Wang, Q., “Tail Attacks on Web Applications,” submitted to CCS ’17.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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)*, 2017.

[13] 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)*, 2017.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

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

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

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] T. Zhang, Y. Zhang, and R. B. Lee, "Memory dos attacks in multi-tenant clouds: Severity and mitigation," *arXiv preprint arXiv:1603.03404*, 2016.

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

[33] H. Shan, Q. Wang, and Q. Yan, "Very short intermittent ddos attacks in an unsaturated system," *submitted to SECURECOMM*, 2017.

[34] H. Shan and Q. Wang, "Tail attacks on web applications," *submitted to CCS*, 2017.

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

[36] 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.

[37] 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.

[38] 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.

[39] 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.

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

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

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

[43] 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.

[44] 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.

[45] 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.

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

[47] 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.

[48] 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.

[49] 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.

[50] 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.

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

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

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

[54] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, “Heracles: improving resource efficiency at scale,” in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, vol. 43, no. 3. ACM, 2015, pp. 450–462.