Tail Amplification in n-Tier Systems

Huasong Shan*, Qingyang Wang*, Shungeng Zhang*, Qiben Yan†
*Louisiana State University, †University of Nebraska-Lincoln

***Abstract*—We present a new type of DDoS Attacks in the cloud, MemCA, with the goal of causing the long-tail latency problem (e.g., 95th percentile response time $>$ 1 second) of the target n-tier web application (e.g., e-commerce) while keeping stealthy. MemCA exploits the sharing nature of public cloud computing platforms by collocating the adversary VMs with the target VMs that deploy the target web application, and causing intermittent and short-lived cross-resource contentions on the target VMs. We show that these short-lived cross-resource contentions can cause transient performance interferences that lead to large response time fluctuations of the target web application because of complex resource dependencies in the system. We further model the attack scenario in n-tier systems based on queuing network theory, and analyze cross-tier queue overflow and tail (response time) amplification under our attacks. Through extensive benchmark experiments in both our private cloud and the public NSFCloud, we confirm that MemCA can cause significant performance degradation of the target n-tier system while keeping stealthy. Specifically, we show that MemCA bypasses not only the cloud elasticity mechanisms, but also the state-of-the-art performance interference detection mechanisms in the cloud.**

I. Introduction

Distributed Denial-of-Service (DDoS) attacks for web applications such as e-commerce are increasing in size, scale, and frequency [1], [15]. On October 21, 2016, a number of popular websites, like Twitter and Spotify, cannot be accessed by some users worldwide, due to two DDoS attacks on the DNS servers of Dyn company [44]. On the other hand, Berkeley's paper "above the clouds: a Berkeley view of Cloud Computing" [13] first introduced cloud computing in 2009 and predicted that the top one opportunity for cloud computing is to use elasticity to defend against DDOS attacks: systems can scale easily to fit the dynamic user requirements, even to serve the attack traffic. Since then a large number of websites are moved into the cloud [22], [53], [57](e.g., Spotify [49] moved its core infrastructure to Google Cloud on Feb. 23, 2016). However, DDoS attacks are still very active and even more severe, because ever-evolving new types of DDoS attacks exploit various newly discovered network or system vulnerabilities even in the cloud, bypassing not only the state-of-the-art defense mechanisms [40], [58], but also the elasticity mechanisms of cloud computing [46], [47], [52].

The new types of DDoS attacks in the cloud can be categorized into two classes: external and internal attacks [14], [19]. External attacks are similar to the traditional DDoS attacks that launch external attacking traffic (both application and network level) to the target services [37], [40], [58], but with different attacking approaches or goals to bypass the scaling capability of the cloud. For example, an external attack may attack third party services (e.g., the Dyn DNS servers outside of the cloud computing platform) that the target services rely on, or cause partial denial of service (DoS) of the target service by sending pulsating but legitimate HTTP traffic to the target system [46], [47]. In contrast, internal attacks are new-born, which are emerging simultaneously with cloud computing and become an important class of DoS attacks due to the resource sharing nature of public cloud [14], [19]. Internal attacks can easily mount the adversary programs in the co-located VMs [45] (on the same host with the target VM) that cause resource contention and performance interference of the target VM and hurt the performance of the target service [21], [59].

Existing approaches to detect and mitigate the performance interference are either provider-centric [38], [60] or user-centric [27], [34], [35]. For provider-centric approaches, cloud providers profile the infrastructure-level metrics from the hosts in the cloud. Due to the requirement of a worthwhile investment of cloud providers (e.g., profiling overhead should be under 1% [29]), cloud providers typically adopt coarse granularity monitoring (in minutes level). For example, the sampling interval of Amazon's monitoring tool CloudWatch [3] and Microsoft Azure Application Insights [39] is typically 1 minute, which is incapable of detecting short-term performance interference (e.g., $<$ 1 minute). For user-centric approaches, cloud tenants protect their applications from performance interference in their rented VMs. They may enable fine-grained monitoring (e.g., 1 second), with the cost of non-trivial monitoring overhead, to detect short-lived performance interference. However, user-centric approaches typically lack host level information (e.g., the state of the co-located adversary VMs), thus are unable to detect the causes of performance interference.

In this paper, we present a new type of internal attacks in the cloud, which can significantly hurt the performance of target web applications in the cloud (e.g., long tail latency) while bypassing the elasticity mechanisms and the interference detection mechanisms in the cloud. Concretely, an attacker mounts an adversary program in the co-located VMs with the target VM that deploys the target web application; the adversary program can degrade the performance of latency-sensitive web systems through creating very-short-lived (less than 1 second) performance interference among the co-located VMs hosted in the same physical machine in the cloud. Such an attack is considered as a great threat for modern web applications that require rapid responsiveness, such as e-commerce, media streaming, and online gaming [1]. At the same

[1]Amazon reported that every 100ms increase in the page load decreases their sales by 1% [30]; Google requires 99 percentage of its queries to finish within 500ms [18].

time, the performance interference caused by the proposed attacks only lasts for very short time period (e.g., 100ms) each time; from the Sampling theory, the average system resource utilization is still in moderate level (e.g., 50%) using coarse granularity monitoring, thus not only avoiding the typical triggering conditions (e.g., CPU usage $>$ 80%) of the cloud scaling, but also escaping the state-of-the-art detection mechanisms of performance interference in the cloud.

To make an effective internal attack, the primary challenge 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 [16], [32], I/O [23], last level cache [59], memory lock [59], and CPU scheduling mechanism of the hypervisors [61]. These shared resources are usually inter-correlated with each other, causing cross-resource contention. For example, network traffic affects last level cache [50] and memory bandwidth affects CPU utilization. These cross-resource contentions [33] make it more difficult to trace the cause of performance interference, which increases the stealthiness of internal attacks. In this paper, we explore one type of cross-resource internal attacks, in which the adversary program intermittently triggers memory resource contention and degrades the available memory bandwidth among the co-located VMs, which transiently saturates the CPU of the target VM. We name this attack *MemCA* (Memory Attacks on the Neighbor's CPU).

In brief, this work makes the following contributions:

- Presenting a new type of internal DDoS Attacks named *MemCA*, that can lead to long-tail latency of web applications with high stealthiness since the average resource utilization of the target servers is far from saturation.
- Introducing a type of cross-resource contentions that can significantly cause performance interference among the co-located VMs in the host inside the cloud.
- Modeling the proposed attack scenario in n-tier systems based on queuing network theory, and analyze cross-tier queue overflow and tail (response time) amplification under our attacks.
- Validating the practicality of our attacks through extensive benchmark experiments in both our private cloud and the public NSFCloud [43], confirming both damage (e.g., 95th percentile response time $>$ 1 second) and stealthiness (bypassing elasticity mechanisms and interference detection mechanisms in the cloud).

We outline the rest of this paper as follows. Section II discusses the background of memory resources and describes our attack scenario and practical impact of tail amplification. Section III investigates two types of memory attacks: saturating memory bus and triggering memory lock. Section IV designs and implements MemCA, models the attack scenario using queuing network theory, and analyzes the attack impacts (e.g., tail amplification). Section V evaluates MemCA from two aspects: damage and stealthiness. Section VI presents related work and Section VII concludes the paper.

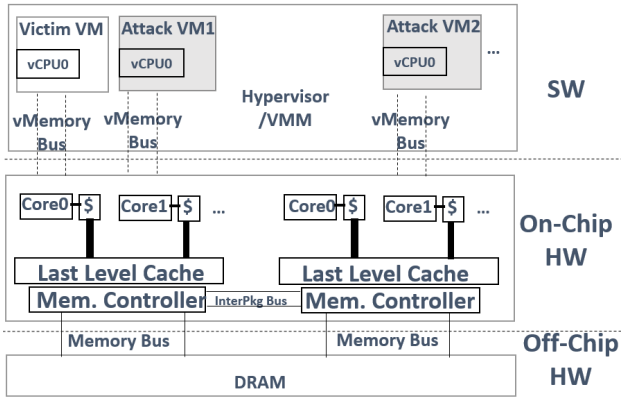


Fig. 1: **On-chip resources shared by co-located VMs.**

II. Background and Motivations

A. Shared On-Chip Resources and Resource Contention

In modern CPU architecture (see Figure 1), such as Intel Xeon family which are widely used by today's IaaS providers, there exists multiple processor sockets splitting on-chip resources into different packages. In the same processor package, last level cache and memory scheduling components (e.g., memory controller bus, bank scheduler, channel scheduler) are shared by co-located VMs except for core-private L1 and L2 caches. The commercial cloud providers can assign specific memory size and CPU cores to customize different instances [4], [6], but it is insufficient to isolate all other on-chip memory resources, leading to significant resource contention between co-located VMs, such as memory bandwidth, even cross-resource contention. For example, RFA [50] investigated performance degradation on LLC caused by network; LLC contention causes co-located VMs to require more memory bandwidth, creating a memory bandwidth contention; memory thrashing (e.g., cache miss in memory systems) presses CPU's scheduling, making CPU contention for application requests.

In this work, we focus on one type of cross-resource contention, specifically, shared on-chip memory resource contention on CPU among the co-located VMs– *MemCA*. This type of cross-resource contention is hard to detect because the cause and the result are indirectly correlated (e.g., CPU saturation does not mean CPU is the bottleneck, but the limited memory bandwidth), which is a big challenge for current detection mechanisms [33]. In addition, typical memory metrics supported by monitoring tools (e.g., sar, collectl) cannot cover all the cases of memory attacks (e.g., memory lock).

B. 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., memory bandwidth) without being detected. Today's software-based VMM (e.g., Xen, KVM, VMware vSphere, Microsoft Hyper-V) can only guarantee secure access to virtual and physical memory pages, but

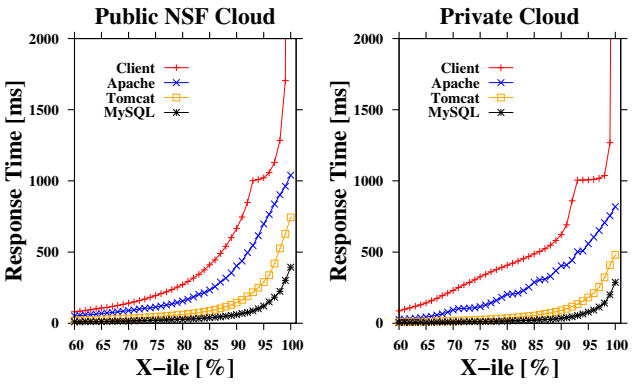


Fig. 2: **Measured tail amplification of response time observed in each tier of a 3-tier system caused by MemCA, and long tail latency observed from clients.**

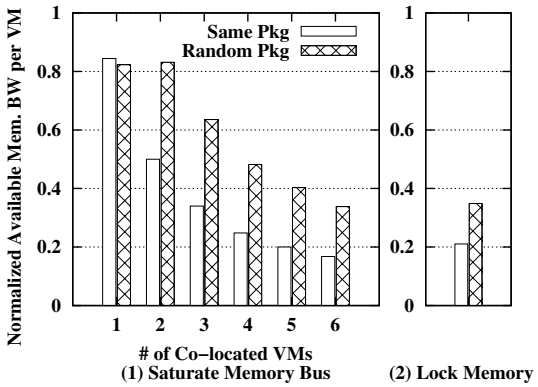


Fig. 3: **Measured memory attack.** (1) As co-located VMs increases, available memory bandwidth used by each VM decreases. (2) Locking memory is more effective in degrading available memory bandwidth than saturating memory bus.

not to isolate those on-chip memory resources shared by the VMs. In this case, the MemCA attackers can increase CPU consumption of the target VM by memory attacks in co-located adversary VMs, even though vCPUs are isolated and protected by the hypervisor. Finally, such attacks are to degrade the quality of service for the target Web systems, especially causing long-tail latency problem in the long run.

To effectively launch a MemCA attack, one precondition is that the adversary VMs should share the same host with the target VMs. Many previous research efforts already provide solutions [45], [51], [57], and are orthogonal to our research. Ristenpart's team [51] reported the cost of achieving co-located VMs in public IaaS cloud (average cost is from $0.137 to $5.304), and the successful rate of co-residency (from 0.6 to 0.89). We also assume that the attacker can fully control the rented adversary VMs, as is the case for today's public IaaS cloud [31].

C. Motivations

Figure 2 shows damage impact caused by MemCA through concrete benchmark web application RUBBoS [8], deployed in both our private cloud and the public NSFCloud [43]. We can note tail amplification of response time observed from each tier in the 3-tier system, in other words, the response time of each tier has a nonlinear tail trend as percentile increases, meanwhile, the tail of each tier amplifies from downstream tier (MySQL) to all upstream tiers (Tomcat and Apache), eventually, causing long tail latency for clients (e.g., 95th percentile response time $>$ 1 second). More experimental details and explanation are available in Section V.

III. Memory Attack Measurements

In this section, we investigate performance impact of memory attacks among the co-located VMs in the representative host widely used by the cloud platform.

Experiment Methodology. In our private cloud, we use 3 machines to build our profiling environment, each of which 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 [4]. Our private cloud is managed by OpenStack Ocata; each VM, running CentOS release 6.7, is managed by KVM [5].

To measure the performance impact of memory attacks among the co-located VMs, we exploit two approaches to launch memory attacks: saturating memory bus through memory benchmarking tools [27], and locking memory access through unaligned atomic operations [59]. At the same time, we measure available memory bandwidth which can be used by each co-located VM as an evaluation metric to assess performance interference caused by memory attacks.

The program to measure memory bandwidth of the host is RAMspeed, a cache and memory benchmarking tool [7]. As we have 6 cores per package in our host, we deploy 6 co-located VMs, each with one vCPU. To comprehensively understand the impact of memory attacks to shared memory bandwidth in modern CPU architecture (see Figure 1), we execute attack programs and measure memory bandwidth in two scenarios:

1) Same package, each of 6 VMs pinned to a separate core on the same package, which share last-level cache and memory bandwidth per package.
2) Random package, 6 VMs floating over 12 cores on two packages, which share last-level cache and memory bandwidth of both packages. This case represents the common practice in real cloud computing platform aiming to increase the level of sharing resources.

Results of Memory Bandwidth Contention among Co-located VMs. Figure 3 depicts measured available memory bandwidth used by each co-located VM in the same and random package under memory attacks through saturating memory bus and locking memory access. We can summarize several conclusions to guide MemCA.

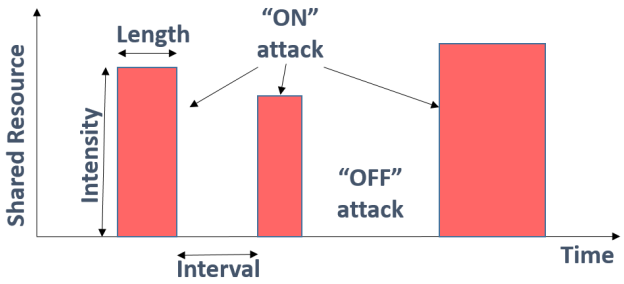


Fig. 4: **An illustration of MemCA bursts**.

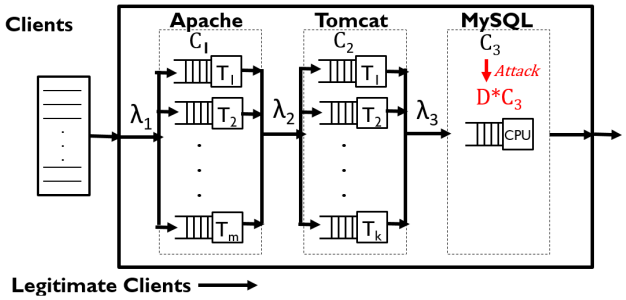


Fig. 5: **MemCA scenario and system model.**

1) One VM running attack program can not saturate memory bus, since memory bandwidth in modern processors may be too high.
2) As co-located VMs increases, available memory bandwidth used by each VM decreases in same package case. In random package case, the trend is similar, but the degradation level is lower, since the capacity of memory bus in different package case may double the one in same package case.
3) Locking memory access is more effective to degrade available memory bandwidth than saturating memory bus since memory access from other applications are completely blocked until the locked action is done.

The previous memory attack experiments are conducted in KVM platform. To further confirm that the effectiveness of these memory attacks in co-located VMs is not related to any specific hypervisor, we also conduct similar experiments in different platforms managed by different popular hypervisors (e.g., Xen, VMware, Hyper-V). We get similar results under the same memory attacks as shown in Figure 3.

IV. MemCA

A. MemCA Overview

Through memory attack experiments in Section III, we profile the capacity of target host and determine the attack force of the adversary program. Past work [59] presented performance impacts of brute force memory attacks in co-located VMs, such attacks can be easily discovered by state-of-the-art detection mechanisms [34], [35], [60]. Therefore, we design an even more stealthy internal attack, MemCA.

The most effective attack approach is to saturate or degrade shared resources (e.g., memory bandwidth, network bandwidth, TCP backlog size) quickly in milliseconds level while bypassing coarse granularity monitoring based detection mechanisms [34], [35], [60] (e.g., second or minute), and intermittently repeat this attack process to cause the long-lasting threat to performance. In other words, MemCA can create very short resource contention bursts (within subsecond) in ON-OFF style (Figure 4). To this end, we formally propose MemCA as follows:

$$Effect = \mathbb{A}(R, L, I) \tag{1}$$

where,

- $Effect$ is the measure of attack impacts, we use percentile response time as a metric to measure tail latency of the target system (e.g., 95ile response time $>$ 1s).
- R is the intensity of resource consumption per interference attack burst. R should be large enough to temporarily saturate or degrade shared resource (e.g., memory bandwidth) in the servers of the cloud.
- L is the lasting period (length) of each interference burst. L should be short enough (e.g., $<$ 1s) to guarantee the interference burst not to be captured by the interference detection mechanisms [34], [35], [60].
- I is the time interval between every two consecutive interference bursts. I infers the frequency of attack bursts. 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 traditional flooding DDoS attacks, which can be easily detected.

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

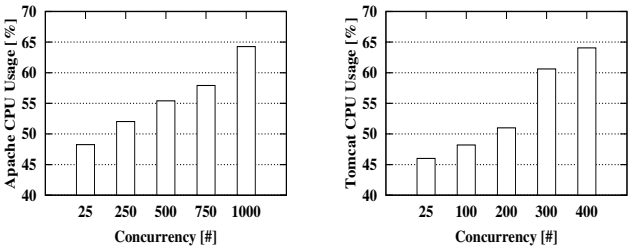
- How can we crystallize the relationship between attack impacts and attack parameters? Section IV-B will use queuing network theory to model the n-tier systems, generalize attack scenarios of MemCA, and analyze the phenomenon of cross-tier queue overflow [55] and response time amplification under MemCA.
- How can the attackers get optimal attack parameters without the knowledge of performance parameters among the n-tier system? Section IV-C will exploit feedback control theory to dynamcally adjust attack parameters for better attack effectiveness and stealthiness, even without knowing the performance parameters of the target systems (service time and resource utilization).

B. MemCA Modeling

System Model and Problem Statement. In queueing theory, a tandem queue is usually adopted to represent a complex computer system, particularly suited for n-tier systems. In n-tier system, the requests are loaded to the first queue, the output of which is fed to the second queue, and so on. Each tier typically has an arrival rate following a Poisson process and a service rate following an exponential distribution, which is

Param.	Description
Q_i	the queue size for the *i*th tier
$C_{i,OFF}$	the capacity for the *i*th tier during OFF bursts
$C_{i,ON}$	the capacity for the *i*th tier during ON bursts
λ_i	the legitimate request rate terminating in the *i*th tier
D	the degradation index of the capacity of the *n*th tier
$l_{i,UP}$	the time to fill up the queue of the *i*th tier per burst
$l_{i,DOWN}$	the time to drain the queue of the *i*th tier per burst
$\tau(i)$	the maximum response time of the *i*th tier in a burst
$P_{i,D}$	the period of $\tau(i)$ of the *i*th tier in a burst
$P_{i,MB}$	the period of a millibottleneck of the *i*th tier in a burst
ρ	overall percentile response time under MemCA

TABLE I: **Model parameters**



(a) CPU of Apache increases as Apache's concurrency increases.

(b) CPU of Tomcat increases as Tomcat's concurrency increases.

Fig. 6: **Server capacity degrades as concurrency increases.**

used by RUBBoS benchmark [8]. In Tandem Queue model, the service rate of each tier is independent among n-tier systems.

To model the target n-tier systems, we also assume the arrival rate terminating in each tier is a Poisson process, and the capacity of each tier is an exponential distribution. Queue size of each tier denotes the size of critical resources of each tier since major portion of the response time of each tier is occupied by or waits for the critical resource of that tier [54]. Figure 5 depicts MemCA scenario and a classic n-tier web applications model based on our RUBBoS environment with 3-tier configuration (e.g, 1 Apache, 1 Tomcat and 1 MySQL). For example, in our RUBBoS experiments, CPU is the critical (bottleneck) resource in MySQL; in Tomcat, a DB connection pool serving to query from MySQL is the critical resource, thus queue size of Tomcat is the size of DB connection pool, here, there are k identical DB connections; Apache and Tomcat are connected by AJP Protocol [24], the AJP connection pool is the critical resource in Apache, thus queue size of Apache is the size of AJP connection pool in Apache, here, there are m identical connectors. Table I summarizes the notation and description of the parameters used in our model.

The purpose of our attacks is to cause the long-tail latency problem while keeping stealthy. To analyze percentile response time of the target n-tier system caused by our attacks, we should pinpoint where is the major time consumed inside the n-tier system. Thus, we analyze the percentile response time observed in each tier. Meanwhile, we also need to quantify the stealthiness of our attacks. In other words, we should quantify the period of each interference burst (millibottleneck).

Capacity Degradation at Concurrency. In MemCA attacks, the attacker can degrade the capacity of the *n*-th tier through adversary programs executed in co-located VMs, blocking the process of requests in the last tier, which leads to queue overflow propagation from the last tier to all upstream tiers. We define D as the degradation index of the capacity of the *n*th tier caused by the adversary programs. In a specific environment, each hardware or software resources in the host should have a peak value, R_{max}. Thus, during each burst the degradation index D is

$$D = \frac{R_{max} - R}{R_{max}} \tag{2}$$

where, R is the attack intensity of each burst.

During ON attack periods, the MemCA attacker quickly saturates the queues in the n-tier system, resulting in each component tier entering the high concurrency state. In a set of RUBBoS experiments in Figure 6 (detailed experimental setting can refer to Section V), we observe that CPU usages of Apache and Tomcat increase as concurrency of the corresponding tier increases under the same throughput in the 3-tier system, indicating capacity degradation of servers as concurrency increases. Thus, we define $C_{i,ON}$ and $C_{i,OFF}$ to differentiate degraded capacity from normal capacity of each tier under our attack scenario.

Similar to Tail Attacks [46], during a MemCA burst the queue of each tier experiences three stages: build up, hold on, and fade off. Thus, we define the degraded capacity of each tier during the three stages of queue state in the following,

$$C_{i,ON} = \begin{cases} [C_{n,ON}, C_{i,OFF}], & \text{build up.} \\ C_{n,ON}, & \text{hold on.} \\ [C_{n,ON}, C_{i,OFF}], & \text{fade off.} \end{cases} \tag{3}$$

where, during hold-on stage the capacity of each tier degrades into the capacity of last tier $C_{n,ON}$ since last tier seems like a gate through which queued requests can leave out. During build-up and fade-off stages, the capacity of each tier in worst situation degrades into the capacity of last tier $C_{n,ON}$.

A MemCA attacker degrades the capacity of the *n*th tier through adversary programs executed in the co-located VMs during build-up and hold-on stage. During fade-off stage, the capacity of last tier restores while the server should serve the queued requests and clear up the queue. Thus, we define the capacity of last tier $C_{n,ON}$ during the three stages as follows,

$$C_{n,ON} = \begin{cases} D * C_{n,OFF}, & \text{build up.} \\ D * C_{n,OFF}, & \text{hold on.} \\ C_{n,OFF}, & \text{fade off.} \end{cases} \tag{4}$$

Quantifying Attack Impacts. To quantify the damage impact of MemCA, we reduce the problem of calculating percentile response time of each tier to the problem of calculating the damage period in a burst. During each ON attack burst, the state of the queue in each tier experiences three stages: build up, hold on, and fade off. We analyze the period of each stage.

During the build-up stage, the queue will be filled up from the last-most tier to each upstream tier, until the front-most tier. *If the queue size satisfies*

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

and the degradation index during build-up stage satisfies

(Condition 2) $\lambda_n > C_{n,ON}$

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

$$l_{n,UP} = \frac{Q_n}{(\lambda_n - C_{n,ON})} \tag{5}$$

$$l_{n-1,UP} = \frac{(Q_{n-1} - Q_n)}{(\lambda_{n-1} + \lambda_n - C_{n,ON})} \tag{6}$$

...

$$l_{1,UP} = \frac{(Q_1 - Q_2)}{(\sum_{i=1}^{n} \lambda_i - C_{n,ON})} \tag{7}$$

where, condition 1 can be held since the system administrator typically configures the queue size of upstream tiers bigger than the queue size of downstream tiers. Condition 2 guarantees the queue of each tier can be built up only if there is enough capacity degradation D of the bottleneck tier.

During the hold-on stage, the queue in each tier keeps full. Once the queue of a tier is full, the biggest damage occurs since every new comming request to the tier will wait for the maximum queueing time. Thus, we can calculate maximum queueing time of each tier during a burst.

$$\tau(\mathrm{i}) = \frac{Q_i}{C_{i,ON}} \tag{8}$$

where, 1/$C_{i,ON}$ is service time of each tier in the condition of all the downsteam tiers with overflown queue.

During the fade-off stage, the queue will drain from the front-most tier to each downstream tier until the last-most tier. During this stage, each tier must serve all the new-coming requests while clearing the queued requests. *If the capacity and the arrival rate of each tier during fade-off stage satisfy*

(Condition 3) $C_{i,ON} > (\lambda_i + ... + \lambda_n)$

for all i=1,...,n, then the time needed to drain the queue for the i-th server during a burst is approximately

$$l_{1,DOWN} = \frac{(Q_1 - Q_2)}{(C_{1,ON} - \sum_{i=1}^{n} \lambda_i)} \tag{9}$$

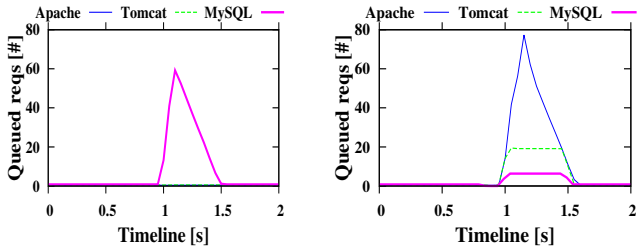
$$l_{2,DOWN} = \frac{(Q_2 - Q_3)}{(C_{2,ON} - \sum_{i=2}^{n} \lambda_i)} \tag{10}$$

...

$$l_{n,DOWN} = \frac{(Q_n)}{(C_{n,ON} - \sum_{i=n}^{n} \lambda_i)} \tag{11}$$

where, condition 3 guarantees the queue of each tier can drain and the fade-off period of each tier can gradually converge only if the capacity of each tier is larger than the arrival rate to that tier.

So far, through analyzing the period of three stages of queue state during a burst, we understand the speed of the queue of each tier build-up and fade-off, and we also know that attack damage impact occurs during the hold-on stage. Thus, we can calculate damage period of the *i*-th tier during a burst, $P_{i,D}$,



(a) In tandem queue, all the requests are queued in last tier. (b) In our attack model, queue overflow propagates in all tiers.

Fig. 7: **Cross-tier queue overflow under MemCA (Figure 7b), comparing to tandem queue model (Figure 7a).**

during which each request in the corresponding tier waits for the maximum queueing time τ(i),

$$P_{i,D} = L - \sum_{k=1}^{i} l_{k,UP} \tag{12}$$

where, L is the burst length during a burst, including build-up and hold-on stage.

In addtion, to guarantee stealthiness of MemCA, we should keep interference bursts (millibottleneck) within subseconds. We reduce the problem of calculating millibottleneck length to the problem of calculating queued length of each tier, including all three stages of queue states for each tier.

$$P_{i,MB} = L + \sum_{k=1}^{i} l_{k,DOWN} \tag{13}$$

where, during fade-off stage, attack is off but queueing time in each tier still matters.

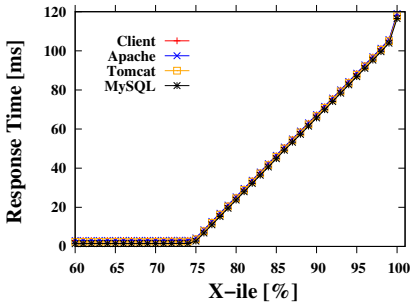
Finally, we can approximately calculate percentile response time of the target n-tier system during the course of an attack,

$$\rho = \frac{P_{1,D}}{I} \tag{14}$$

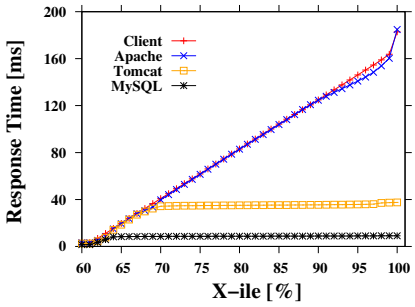
where, I is the burst interval during a burst, including ON and OFF attack, $P_{1,D}$ is damage period of the front-most tier.

Simulation Setting. To further validate the proposed model and quantify attack impacts, we adopt the simulation, Java Model Tools (JMT) [17]. 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 very suitable to evaluate the impact of our attacks in n-tier systems. We modify the JMT code and define the degradation index of service rate during a burst to simulate the bursts of internal attacks. To simplify, we fix burst interval I and burst length L as 2 seconds and 100 milliseconds. With regard to the system configuration in JMT experiments, we estimate constant parameters (λ, C) from our RUBBoS environments [46].

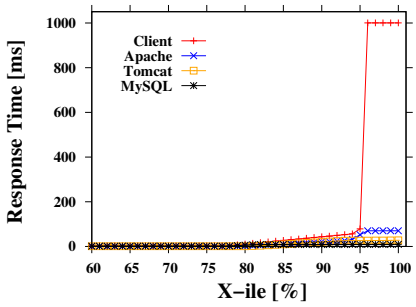
Cross-Tier Queue Overflow. In Figure 7, we compare queue variation of our system model with the one of a tandem queue under the same internal attack case (e.g., D = 0.1). Obviously, Figure 7b shows the process of queue amplification



(a) In tandem queue case with infinite MySQL queue, percentile response time observed by all the tiers and client nearly overlap and continuously increase due to increased queueing time in MySQL. All the requests are queued in MySQL under attacks (see Figure 7a).



(b) In our attack model case with infinite queue in Apache but finite queue in other tiers, response time of each tier amplifies due to cross-tier queue overflow and capacity degradtion of each tier at concurrency, eventually client perceives longer peak response time than that in Figure 8a.



(c) In our attack model case with finite queue of each tier, client perceives longer peak response time than that in Figure 8b, since requests are dropped once all the queues in n-tier system are full, leading to TCP retransmission (the minimum time out of TCP retransmission is 1 second).

Fig. 8: **Tail Amplification under MemCA.** Figure 8a and Figure 8b show the comparison between the tandem queue case and our model with infinite Apache queue. Figure 8c shows more severe response time amplification in the case with finite queues widely used in real production n-tier systems.

in our system model and model analysis, involving queue fill-up, hold-on, and fade-off, since new coming requests will be queued in upstream tier once the queue of its adjacent downstream tier is full. On the contrary, in the tandem queue case in Figure 7a, all the requests are queued in the last tier.

Tail (Response Time) Amplification. In Figure 8, we use the same attack parameters (e.g., D = 0.1, I = 2s, T = 100ms) to launch internal attacks, compare three cases, including a tandem queue model, our attack model with infinite Apache queue (not drop requests), and our attack model with finite Apache queue (can drop requests once all queues are full). Figure 8a shows tandem queue case with infinite MySQL queue, percentile response time observed by all the tiers and client nearly overlap and continuously increase due to increased queueing time in MySQL. All the requests are queued in MySQL under attacks (see Figure 7a). Figure 8b shows our attack model case with infinite Apache queue, response time of each tier amplifies due to cross-tier queue overflow and capacity degradation of each tier at concurrency, eventually, client perceives longer peak response time than that in Figure 8a. Figure 8c shows our attack model case with the finite queue of each tier, client perceives longer peak response time than that in Figure 8b, since requests are dropped once all the queues in n-tier system are full, leading to TCP retransmission (the minimum time out of TCP retransmission is 1 second).

Relationship between Attack Impacts and Attack Parameters. MemCA has two attack goals, damage (e.g., 95th percentile response time $>$ 1s) and stealthiness (e.g., interference length or millibottleneck length $<$ 1s). Through equation 13 and 14, we can calculate percentile response time and millibottleneck length if we know system parameters and attack parameters. On the contrary, based on the predefined attack goals, we also can calculate attack parameters if we know system parameters.

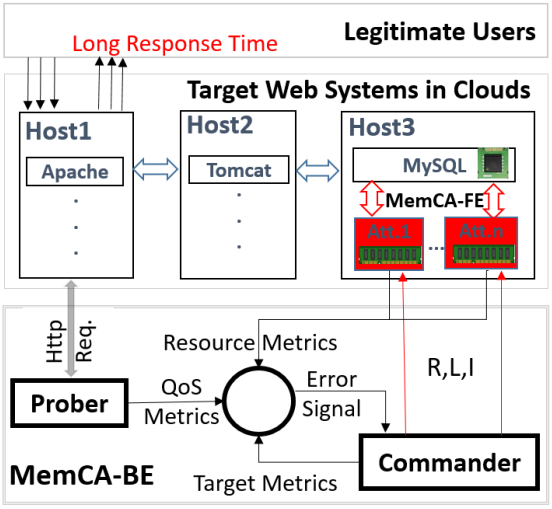


Fig. 9: **MemCA control framework and an 3-tier sample topology.**

C. MemCA Implementation

It is hard to accurately know performance parameters of the target n-tier system for the attackers. However, given the proposed model and analysis, we understand the relationship between attack parameters and their impacts. So we exploit an advanced feedback control tool, Kalman filter [28], to dynamically tune attack parameters to fit shifty system state [46]. We implement an control framework including two components: MemCA front end (MemCA-FE) and MemCA back end (MemCA-BE) (Figure 9). MemCA-FE executes attack programs in co-located adversary VMs and reports the consumption of the shared resource. MemCA-BE consists of two entities: the prober which periodically sends HTTP requests to the target Web systems and monitors performance

metrics of the target web applications, and the commander which dynamically controls attack parameters in attack VMs based on the performance and resource metrics of historical attack bursts.

Estimate Critical Resource Utilization (MemCA's damage). In MemCA-FE, we record the critical resource utilization consumed by the attack VMs through attack program (e.g., RAMspeed). In our attack case, critical resource is memory bandwidth of the physical machine hosting both attack VMs and the target VM. We can profile maximum memory bandwidth of the target machine with a specific hardware.

In our attack control framework, we treat the n-tier system as a black-box system and measure percentile response time of target web application through prober in MemCA-BE. Due to the positive relationship between response time and resource utilization, we can tune the attack intensity R using feedback control technology, to control the target resource utilization. Through adjusting the burst length L and the burst interval I, we can achieve an expected attack damage goal (e.g., 95th percentile longer than 1 second).

Estimate Interference Length (MemCA's stealthiness). We record execution time of attack program (e.g., the RAMspeed benchmark) in the attack VMs in MemCA-FE. During execution time of attack program, the target bottleneck resource is probably busy and saturated. Thus, we use execution time as the millibottleneck length of the critical resource, which can be used to help adjust attack parameters (e.g., burst length L) to achieve the predefined attack stealthy goal. In practice, this approach is very conservative, during execution time of attack program there exist many tiny time slots during which the bottleneck resource can be idle. The actual attack can be more stealthy than the measured value using execution time to estimate millibottleneck length.

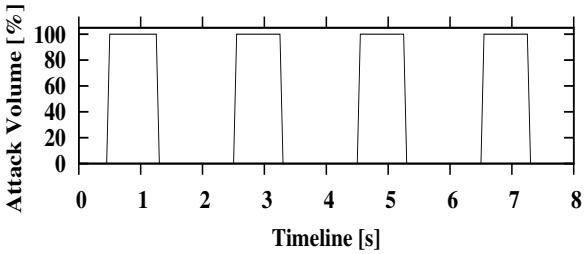
V. MemCA Evaluation

Here, we use the RUBBoS benchmark web applications to evaluate damage and stealthiness of MemCA attacks.

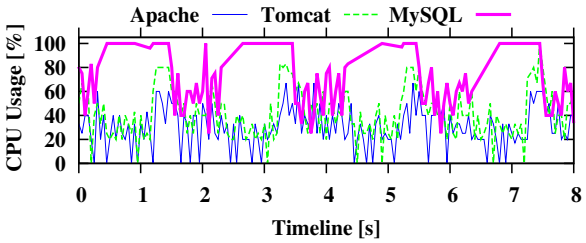
A. MemCA Damage in RUBBoS

Experiment Methodology. We adopt RUBBoS [8], a representative n-tier web application benchmark modeled after the popular news website Slashdot. We configure RUBBoS using the typical 3-tier architecture (see target web systems in clouds in Figure 9) in our private cloud, each tier is deployed in a VM hosted by a dedicated physical machine (Host specification can refer to Section III). We configure 3000 concurrent legitimate users by RUBBoS clients to interact with the target benchmark website. Each user follows a Markov chain model to navigate among different web pages, with averagely 7-second think time between every two consecutive requests.

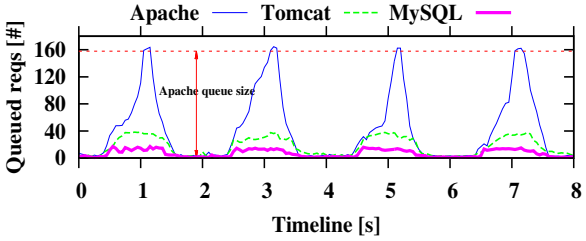
The adversary VMs are co-located with MySQL VM. As for attack parameters, we fix burst interval I as 2 seconds, burst length L as 700 milliseconds, in which average utilization is moderate, bypassing the state-of-the-art detection of performance interference (based on second level monitoring). Given the knowledge in Section III, we saturate the memory bandwidth of the target host through triggering memory lock during each attack burst, which is more effective than saturating memory bus. We predefine the attack goal is to cause long tail latency (e.g., 95% percentile response time $>$ 1 second).



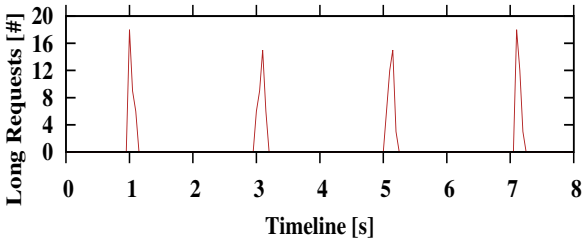
(a) 4 attack bursts lauched by the co-located attack VM during 8-second experimental period. Each burst lasts 700ms, triggering memory lock by the adversary program running in the attack VM to degrade available memory bandwidth in the host.



(b) Transient CPU saturations of the target MySQL VM casued by the attack bursts in the co-located attack VM.



(c) Quickly queue propagation in the 3-tier system during each burst.



(d) Very long response time (e.g., $>$ 1 second) perceived by the end users due to MemCA bursts.

Fig. 10: MemCA Damage in RUBBoS.

Results. We have conducted 3-minute RUBBoS experiments under MemCA attacks launched by our control framework. Figure 2 shows the corresponding percentile response time observed from each tier. Our attack achieved the attack goal. Figure 10 captures 8-second representative snapshot which shows 4 attack bursts to explain the fundamental reason why MemCA attack burst can cause long response time. To observe

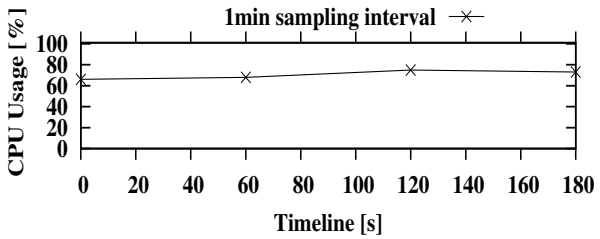
the pattern of corresponding metrics and their correlation, we use 50ms monitoring granularity during the experiment. Figure 10a shows 4 bursts during 8-second attacks. Each burst lasts 700ms, triggering memory lock by our adversary program running in an attack VM to degrade the available memory bandwidth in the host. The attack bursts in the attack VM cause transient CPU saturations of MySQL deployed in the co-located VM in Figure 10b. These transient CPU saturations create millibottleneck (less than 1 second) and cause requests to queue in MySQL; MySQL local queue soon is filled up during each burst, pushing requests to queue in upstream Tomcat and Apache in Figure 10c. Once the queued requests in the front-most Apache exceed its queue limit, new requests from legitimate users will be dropped, leading to TCP retransmissions. The minimum timeout expiration of TCP retransmission [25] is 1 second. Thus, end users perceive very long response time as we observed in Figure 10d.

B. MemCA Stealthiness under Cloud Elasticity and Cloud Detection of Performance Interference

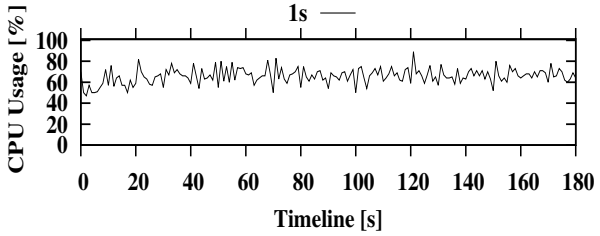
Experiment Methodology to Evaluate MemCA under Cloud Elasticity. Amazon provides Elastic Load Balancing in its cloud platform AWS to guarantee scalability, performance, and security for any application [12]. It can elastically scale to fit the demands of the customers due to the ability to trigger Auto Scaling [2] in the available Amazon EC2 instance fleet. The trigger mechanism of Auto Scaling is based on coarse granularity monitoring tool, Amazon CloudWatch [3], which sampling period is 1 minute. For example, the customers' application can scale out 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 requires verifying whether our attack can offend the threshold of Auto Scaling. In our experiments, we assume the preconfigured threshold of scaling out more instances is 85% of the average CPU utilization, which is usually a feasible and simple solution for system administrators [9], [10].

Results. Figure 11 depicts CPU uitilization of the bottleneck tier MySQL in the previous 3-minute MemCA attacks experiments using different sampling granularity. Figure 11a, 11b, and 11c respectively utilize 1-minute, 1-second, and 50-millisecond 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 granularity 50 milliseconds, we can observe the saturation of CPU utilization. These peek value occurs frequently but transiently, thus the average value is moderate from Sampling theory as shown in Figure 11a and 11b. That's the essential reason why MemCA attacks can bypass the state-of-the-art cloud elasticity mechanisms.

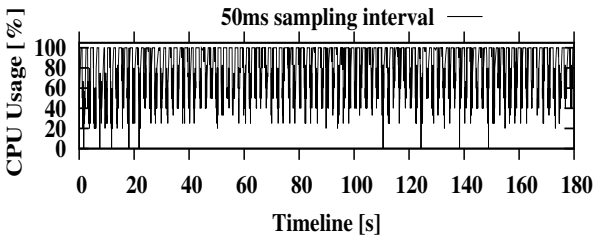
Experiment Methodology to Evaluate MemCA under Cloud Detection of Performance Interference. The sources of performance interference inside the cloud typically are



(a) Using 1-minute monitoring, observing flat CPU usage.



(b) Using 1-second monitoring, showing a little bit fluctuation.

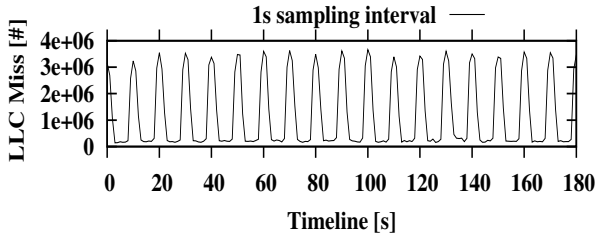


(c) Using 50-millisecond monitoring, observing frequent and transient saturations of CPU uitilization.

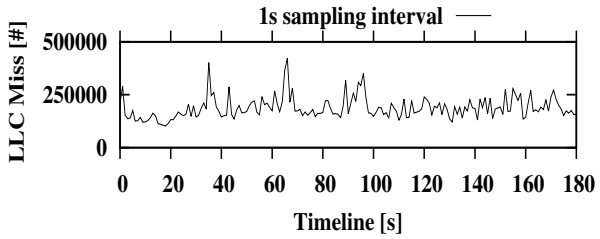
Fig. 11: MemCA Stealthiness under Cloud Elasticity.

low-level metrics (e.g., hardware performance counts), which can not be measured from the customers' VMs, such as cache miss, memory bandwidth, etc. Thus, most detection approaches of performance interference are based on low-level monitoring tools, such as Xentrace [36], OProfile [11]. Here, we use OProfile to measure LLC cache miss, since we try to intermittently degrade memory bandwidth or 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, we get similar results as the observation of CPU usage of MySQL in Figure 11.

Results. Figure 12 shows LLC cache miss of physical machine hosting MySQL and its co-located VMs (PM3 in Figure 9) using two measured memory attack approaches (saturating memory bus and trigger memory lock) in Section III in MemCA experiments of RUBBoS environments.Due to on-off pulsating attack styles of MemCA, we can observe periodic LLC miss in MySQL VM in Figure 12a, which is caused by intermittently saturating memory bus in co-located attack VMs. However, when we using the approach of trigger memory lock in co-located VMs to launch MemCA, we can not observe the obvious pattern of LLC miss in MySQL VM as shown in Figure 12b, even though we create periodic attack



(a) Periodic LLC miss in MySQL VM due to attack bursts of staturating memory bus in co-located VMs.



(b) No obvious pattern of LLC miss in MySQL VM under periodic attack bursts by locking memory in co-located VMs.

Fig. 12: MemCA Stealthiness under Cloud Detection of Performance Interference.

bursts as shown in Figure 10a.

Remarks about MemCA Stealthiness. To sum up, we get the same conclusion that performance interference length is more important for the detection mechism [56], rather than the peak value. In MemCA which causes long tail latency through triggering millibottleneck in n-tier systems, we guarantee performance interference length within subsecond, thus, average utilization of the target systems is moderate, bypassing and invalidating coarse monitoring based cloud scaling mechanisms and cloud detection mechanisms of performance interference as shown in Figure 11.

Moreover, MemCA exploits memory attacks in adversary VMs transiently saturating CPU of the co-located target VM. This type of cross-resource interference [33] is even harder for system administrators to trace and detect the root cause of performance interference. If we use memory lock, we cannot capture the attack pattern even using low-level monitoring tool as shown in Figure 12. In addition, MemCA is launched inside the cloud, leaving no trace like external HTTP requests.

VI. Related Work

In this section, we review the most relevant work with regard to memory performance attacks and its corresponding solutions in today's cloud platforms.

Memory Performance Attacks. Memory is a critical bridge in modern computer architecture linking the high-speed device (e.g., CPU) with the low-speed device (e.g., disk), a well designed memory framework can smoothly balance the workload and significantly speed up the whole performance of the system. On the contrary, memory is also a computer resource that is most frequently exploited by the attacker to degrade the performance of the target system. The traditional memory performance attacks in multi-core systems [41] will fail on modern hardware and cloud platforms. Zhang et al. [59] exploit two types of memory contentions: 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. [26] attack memory bus/bandwidth of a mobile device by an 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 impacted resource is CPU while the attack target is memory) is harder to detect by the defender than the previous single-resource memory attacks. More strikingly, our attacks target n-tier systems, in which there typically exists strong dependency among the distributed nodes [46], the damage impact of our attacks (long-tail latency problem) can be even stealthy and severe, due to the tail amplification.

Cross-Resource Interference. Mitigating single-resource interference in the cloud has been extensively investigated [20], [38], [42], [60]. Cross-resource interactions (e.g., MemCA) is a more difficult problem than single resource interference. Resource freeing attack [50] is a 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 [33] integrated multiple isolation schemes to solve the problem of cross-resource interferences, but it requires a complex design because the number of possible interactions exponentially increases with the number of interference sources. Finally, 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 challenging on the modern CPU architecture [48]. In fact, MemCA (high interference and slow latency) makes the final goal of data centers even harder to achieve.

VII. Conclusion

In this paper, we described MemCA Attacks, a new type of low-volume internal DoS attacks in the cloud. Such attacks exploit the sharing nature of public cloud computing platforms by collocating the adversary VMs with the target VMs that deploy response time sensitive web applications. We showed that MemCA is able to cause intermittent and short-lived cross-resource contentions that lead to large response time fluctuations of the target web application (Section III). We modeled the attack scenario in n-tier systems based on queuing network theory and analyzed cross-tier queue overflow and response time amplification under MemCA (Section IV-B). To validate the practicality of our attacks, we evaluated our attacks through extensive benchmark experiments in both our private cloud and the public NSFCloud (Section V), and confirmed the significant damage and the high stealthiness of our attacks. In general, MemCA Attacks make an important contribution towards understanding the emerging low-volume DDoS attacks in the cloud era.

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

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

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

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

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

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

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

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

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

[11] Oprofile. "http://oprofile.sourceforge.net/".

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

[13] 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, Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[14] 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)*, pages 260–264. IEEE, 2010.

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

[16] 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*, pages 463–469. ACM, 2012.

[17] 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)*, pages 119–120. IEEE, 2006.

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

[19] 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)*, pages 67–71. IEEE, 2013.

[20] C. Delimitrou and C. Kozyrakis. Paragon: Qos-aware scheduling for heterogeneous datacenters. In *ACM SIGPLAN Notices*, volume 48, pages 77–88. ACM, 2013.

[21] 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*, pages 599–613. ACM, 2017.

[22] 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*, pages 177–190. ACM, 2013.

[23] Q. Huang and P. P. Lee. An experimental study of cascading performance interference in a virtualized environment. *ACM SIGMETRICS Performance Evaluation Review*, 40(4):43–52, 2013.

[24] IETF. *RFC 6298*. "https://tomcat.apache.org/connectors-doc/ajp/ajpv13a.html".

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

[26] 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*, pages 716–727. ACM, 2017.

[27] 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)*, pages 135–144. IEEE, 2017.

[28] R. E. Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[29] 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*, page 51. IEEE Computer Society Press, 2012.

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

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

[32] 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*, pages 65–76. ACM, 2010.

[33] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis. Heracles: improving resource efficiency at scale. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 450–462. ACM, 2015.

[34] 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)*, pages 91–100. IEEE, 2015.

[35] 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*, pages 277–288. ACM, 2014.

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

[37] 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*, 7(2-4):216–239, 2015.

[38] 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*, pages 248–259. ACM, 2011.

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

[40] J. Mirkovic and P. Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

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

[42] 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*, number EPFL-CONF-185984, 2013.

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

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

[45] 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*, pages 199–212. ACM, 2009.

[46] 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.

[47] 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.

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

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

[50] 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*, pages 281–292. ACM, 2012.

[51] 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*, pages 913–928, 2015.

[52] 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*, pages 1530–1541. ACM, 2015.

[53] 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*, pages 101–114. ACM, 2014.

[54] 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)*, pages 31–40. IEEE, 2013.

[55] 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)*, pages 207–217. IEEE, 2017.

[56] 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)*, volume 13, pages 329–342, 2013.

[57] 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*, pages 929–944, 2015.

[58] 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*, 15(4):2046–2069, 2013.

[59] 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)*, pages 253–265. ACM, 2017.

[60] 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*, pages 379–391. ACM, 2013.

[61] F. Zhou, M. Goel, P. Desnoyers, and R. Sundaram. Scheduler vulnerabilities and coordinated attacks in cloud computing. *Journal of Computer Security*, 21(4):533–559, 2013.