Tail Attacks on Web Applications

Huasong Shan[†], Qingyang Wang[†], Calton Pu[*]

[†]*Computer Science and Engineering Division, Louisiana State University, Baton Rouge, LA, USA*
{hshan1, qwang26}@lsu.edu

[*]*College of Computing, Georgia Institute of Technology, Atlanta, GA, USA*
calton@cc.gatech.edu

ABSTRACT

As the extension of Distributed Denial-of-Service (DDoS) attacks to application layer in recent years, researchers pay much interest in these new variants due to a low-volume and intermittent pattern with a higher level of stealthiness, invaliding the state-of-the-art DDoS detection/defense mechanisms. We describe a new type of low-volume application layer DDoS attack–Tail Attacks on Web Applications. Such attack exploits a newly identified system vulnerability of n-tier web applications (millibottlenecks with sub-second duration and resource contention with strong dependencies among distributed nodes) with the goal of causing the long-tail latency problem of the target web application (e.g., 95th percentile response time > 1 second) and damaging the long-term business of the service provider, while all the system resources are far from saturation, making it difficult to trace the cause of performance degradation.

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

CCS CONCEPTS

• **Security and privacy → Distributed systems security**; **Web application security**; **Denial-of-service attacks**;

KEYWORDS

Long-tail latency; milli-bottleneck; n-tier systems; pulsating attack; web attack; DDoS attack

CCS '17, October 30-November 3, 2017, Dallas, TX, USA
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-4946-8/17/10…$15.00
https://doi.org/10.1145/3133956.3133968

1 INTRODUCTION

Distributed Denial-of-Service (DDoS) attacks for web applications such as e-commerce are increasing in size, scale and frequency [1, 5]. 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. Web applications remain the most vulnerable entrance for any enterprise and organization, so the attackers can exploit them to launch both low-volume and stealthy application-layer DDoS attacks. On the other hand, in web applications especially e-commerce websites, fast response time is critical for service providers' business. For example, Amazon reported that an every 100ms increase in the page load is correlated to a decrease in sales by 1% [24]; Google requires 99 percentage of its queries to finish within 500ms [10]. Emerging augmented-reality devices (e.g., Google Glass) need the associated web applications with even greater responsiveness in order to guarantee smooth and natural interactivity. In practice, the tail latency, rather than the average latency, is of particular concern for response-time sensitive web-facing applications [6, 10, 11, 18, 40].

In this paper we present a new low-volume application-layer DDoS attack–Tail Attacks, significantly worsening the tail latency on web applications. Web applications typically adopt n-tier architecture in which presentation (e.g., Apache), application processing (e.g., Tomcat), and data management (e.g., MySQL) are physically separated among distributed nodes. Previous research on performance bottlenecks in n-tier systems [41–43] shows that very short bottlenecks (VSBs) or millibottlenecks (with sub-second duration) with dependencies among distributed nodes not only cause queuing delay in local tier, but also cause significant queuing delay in upstream tiers in the invocation chain, which will eventually cause the long-tail latency problem of the target system (e.g., 95th percentile response time > 1 second). More importantly, this phenomenon usually starts to appear under moderate average resource utilization (e.g., 50%) of all participating nodes, making it difficult to trace the cause of performance degradation. In the scenario of Tail Attacks, an attacker sends intermittent bursts of legitimate HTTP requests to the target web system, with the purpose of triggering millibottlenecks and cross-tier queue overflow, creating "Unsaturated DoS" and the long-tail latency problem, where denial of service can be successful for short periods of time (usually tens or hundreds of milliseconds), which will eventually damage the target website's reputation and business in the long term.

The study of Tail Attacks complements previous research on low-rate network-layer DDoS attacks [12, 14, 21, 22, 25, 27, 39],

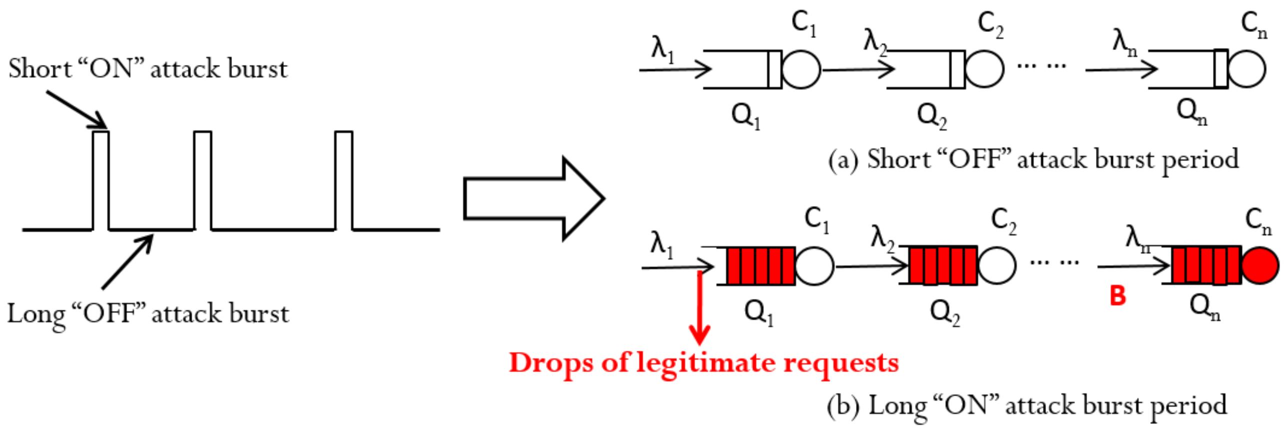

Short “ON” attack burst
Long “OFF” attack burst
λ1
C1
Q1
λ2
C2
Q2
λn
Cn
Qn
(a) Short “OFF” attack burst period
B
Drops of legitimate requests
(b) Long “ON” attack burst period

Figure 1: Attack scenario and system model

low-volume application-layer DoS Attacks [8, 28], and flash crowds (usually tens of seconds or minutes) [19, 38, 44] which refer to the situation when thousands of legitimate users suddenly start to visit a website during tens of seconds or minutes due to a flash event (e.g., during the week of Thanksgiving). The uniqueness of Tail Attacks from previous research is that Tail Attacks aim to create very short (hundreds of milliseconds) resource contention (e.g., CPU or disk I/O) with dependencies among distributed nodes, while giving an “Unsaturated illusion" for the state-of-the-art IDS/IPS tools leading to a higher level of stealthiness.

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

In brief, this work makes the following contributions:

- Proposing Tail Attacks by exploiting resource contention with dependencies among distributed nodes, that can significantly cause the long-tail latency problem in web applications while the servers are far from saturation.
- Modeling the impact of our attacks on n-tier systems based-on queueing network theory, which can effectively guide our attacks in an even stealthy way.
- Adopting a feedback control-theoretic (e.g., Kalman filter) framework that allows our attacks to fit the dynamics of background requests and system state by dynamically tuning the optimal attack parameters.
- Validating the practicality of our attacks through not only analytical, numerical, and simulation results but also real experimental results of a representative benchmark website equipped with state-of-the-art DDoS defense tools in real cloud production settings.
- Presenting a conceptual solution to detect and defense the proposed attacks, involving three stages: fine-grained monitoring, identifying bursts, and blocking bots.

We outline the rest of this paper as follows. Section 2 describes an attack scenario and the practical impact of Tail Attacks in real cloud production settings. Section 3 models our attack scenarios in an n-tier system using queueing network theory, and provides an effective approach to solve the potential optimal attack parameters numerically. Further, we evaluate the attack analytical model in JMT [7] simulator environment and suggest several guidelines to choose the optimal attack parameters in more complex cases (e.g., the competition for free slots of a queue between attack requests and normal requests, overloaded attack requests can be also dropped by the front-tier server). Section 4 describes our concrete implementation to launch Tail Attacks in real web applications. We adopt Kalman filter, a feedback control-theoretic tool, to automatically adjust the optimal attack parameters fitting the dynamics of target system state(e.g., dataset size change) and background workload. Section 5 shows our attack results of RUBBoS [34] benchmark website we have conducted in real cloud production settings, which further confirm the effectiveness and stealthiness of the proposed attacks, and the practicality of the control framework in more practical Web environments. Section 6 provides a “tit-for-tat" strategy to detect and defend our attacks targeting the unique scenario and feature of the proposed attack. Section 7 discusses some additional factors that may impact the effectiveness of Tail attacks. Section 8 presents the related work and Section 9 concludes the paper.

2 SCENARIO AND MOTIVATIONS

Attack Scenario. Consider a scenario of a Tail attack in an n-tier system in Figure 1. The detailed model analysis and experimental

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

W/o att.: Without attacks, **Att.**: under Tail Attacks, **RT**: response time(ms)

Table 1: The measured long-tail latency under Tail Attacks in real cloud production setting.

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

Table 2: Model parameters

data of Tail Attacks are in the following sections. By alternating short “ON" and long “OFF" attack burst, an attacker guarantees the attack both harmful and stealthy. Short “ON" attack burst is typically on the order of milliseconds. The following sequence of causal events will lead to the long-tail problem at moderate average utilization levels during the course of Tail Attacks. (**Event1**) The attackers send intermittent bursts of attack but legitimate HTTP requests to the target system during the “ON" burst period; each burst of attack requests are sent out within a very short time period (e.g., 50ms). (**Event2**) Resource millibottlenecks occur in some node, for example, CPU or I/O saturates for a fraction of a second due to the burst of attack requests. (**Event3**) A millibottleneck stops the saturated tier processing for a short time (order of milliseconds), leading to fill up the message queues and thread pools of the bottleneck tier, and quickly propagating the queue overflow to all the upstream tiers of the n-tier system as shown in Figure 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 [16]), the end users with the requests encountering TCP retransmissions perceive very long response time (order of seconds).

Long “OFF" attack burst is typically on the order of seconds, in which the target system can cool down, clearing up the queued requests and returning back to a low occupied state shown in Figure 1a. Unlike the traditional flooding DDoS attacks which aim to bring down the system, our attack aims to degrade the quality of service by causing the long-tail latency problem for some legitimate users while keeping the attack highly stealthy. The alternating short “ON" and long “OFF" attack burst can effectively balance the trade-off between attack damage and elusiveness.

Measured Long-Tail Latency. Table 1 shows the impact of Tail Attacks through concrete benchmark web application with real production settings deployed in the most popular two commercial cloud platforms (Amazon EC2 [3], Microsoft Azure [29]) and one academic cloud platform (NSF Cloudlab[31]). We use a notation CloudPlatform-ServerTiers-BaselineWorkload to denote the cloud platform, the configuration of the n-tier system, and the background workload. For Server Tiers, We use a four-digit (or three-digit) notation #W#A#L#D to denote the number of web servers, application servers, load-balance servers (may not be configured), and database servers. More experimental details are available in Section 5.1. Table 1 compares the tail latency of the target system under attack and without attack, indicating the significant long tail latency problem under attack. Such long tail latency problem (e.g., 95th percentile response time > 1 second) is considered as severe performance degradation by most modern e-commerce web applications (e.g., Amazon) [6, 10, 11, 18, 24]. At the same time, the average response time is still in acceptable range under attack, making the illusion of “business as usual" for system administrators.

3 TAIL ATTACKS MODELING

In this section, we provide a simple model to analyze the impact of our attacks to the end-users and the victim n-tier system. Based-on the simplified model we introduce an effective approach of getting the potential optimized attack parameters to achieve our attack goal. Finally, we evaluate the model via simulation experiments and suggest several approaches to tune the attack parameters.

3.1 Model

Queueing network models are commonly used to analyze the performance problems in complex computer systems [23], especially for performance sizing and capacity provision. Here, we use a well-tuned queuing network to model n-tier systems, and analyze the sequence of causal events and the impact in the context of Tail Attacks shown in Figure 1. Table 2 summarizes the notation and description of the parameters used in our model. The basic attack pattern (see Event1 in Section 2) shown in Figure 1 is that during the “ON" burst period (L) the attackers send a burst of attack requests with the rate (B), after the “OFF" burst period (T-L) they send another burst again, and repeat this process during the course of a Tail attack. If all the attack requests will not be dropped by the target system (more complex case will be discussed in Section 3.3), we can calculate the attack volume during a burst by:

$$V = B * L \tag{1}$$

We assume that the external burst of legitimate HTTP requests (Event1 in Section 2) can cause sudden jump of resource demand flowing into the target system and cause millibottlenecks (Event2 in Section 2) in the weakest point of the system [12]. 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) caused by overloaded attack bursts can lead to cross tier queue overflow from downstream tiers to upstream tiers (Event3 in Section 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

$$l_n = \frac{Q_n}{(\lambda_n + B - C_{n,A})} \tag{2}$$

$$l_{n-1} = \frac{(Q_{n-1} - Q_n)}{(\lambda_{n-1} + \lambda_n + B - C_{n,A})} \tag{3}$$

$$\cdots$$

$$l_1 = \frac{(Q_1 - Q_2)}{(\sum_{i=1}^{n} \lambda_i + B - C_{n,A})} \tag{4}$$

When millibottlenecks occur in the *n*-th server, firstly the queue in the *n*-th tier is overflown during l_n, which equals the available queue size of the *n*-th tier divided by the newly-occupied rate for the queue of the *n*-th tier in Equation 2. The available queue size equals the queue size of each tier subtracting the queue size of its directly downstream tier. The newly-occupied rate equals the incoming rate of each tier subtracting the outgoing rate of the total system ($C_{n,A}$), the incoming rate of the *n*-th tier includes the requests going through the *n*-th tier and terminating in the *n*-th tier (B for the attack requests and λ_n for the normal requests). We carefully choose the attack requests [40] guaranteeing the attack requests go through every tier and terminate in the last bottleneck tier (detailed implementation in Section 4.2). Equation 3 represents the time to fill up the queue in the *n-1*-th tier. Because one queued request in a downstream server holds a position in the queue of every upstream server, after the *n*-th tier is full, the available queue size of the *n-1*-th tier should be (Q_{n-1} - Q_n). All the requests arriving to a downstream tier need to go through every upstream tier, thus the incoming rate of the *n-1*-th tier includes the request rate of terminating in the *n-1*-th tier (λ_{n-1}) and the request rate of going through the *n-1*-th tier (λ_n + B). Similarly, we can calculate the time to fill up every queue in the n-tier system during the process of propagating the queue overflown. Finally, the required time to overflow all the queues in the n-tier system is the sum of l_i. Here, Q, C, and λ are constants. l_i is a function of *B*.

Once all the queues are overflown (Event3 in Section 2) in the n-tier system, the new incoming requests may be dropped by the front tier (Event4 in Section 2). We term the period of the requests dropped during a burst as *damage length*. If the attackers continue to send attack requests to the system with overflown queues, and we assume that attack requests can always occupy the free position of the queue in the system (more complicated case will be discussed in Section 3.3), then we can approximately infer *damage length* by:

$$P_D = L - \sum_{i=1}^{n} l_i \tag{5}$$

Further, the end-users with the dropped requests perceive very long response time (Event5 in Section 2), leading to the long-tail latency problem caused by our attacks (Event1 in Section 2), which can be approximately estimated as follows,

$$\rho(\mathrm{T}) = \frac{P_D}{T} \tag{6}$$

During a burst, the servers need to provide all the required computing resources (including bottleneck resources) to serve both attack requests and normal requests. We term the period of a millibottleneck during a burst as *millibottleneck length*, during which bottleneck resources sustain saturation. Thus, *millibottleneck length* should involve the resource consumption for both attack and normal requests during a burst. Equation (7) represents *millibottleneck length* derived through the geometric progression in mathematics (more detailed derivation of this equation in Appendix A).

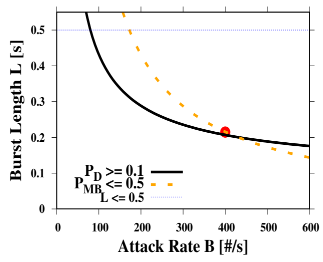
$$P_{MB} = V * \frac{1}{C_{n,A}} * \frac{1}{(1 - (\lambda_n * \frac{1}{C_{n,L}}))} \tag{7}$$

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

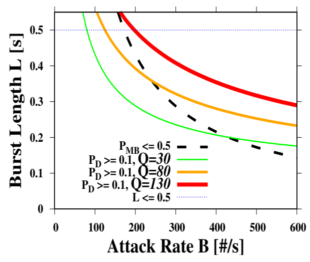
3.2 Numerically Solve Attack Parameters

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

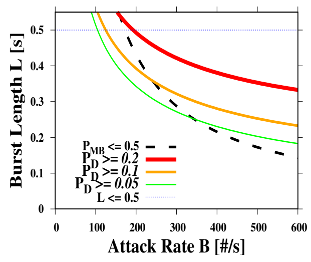
Constant Parameters Estimation. To get some reasonable constant parameters(λ,$C_{i,A}$,$C_{i,N}$,Q) in the model, we estimate these constants via profiling the service time of each type of request of each component tier in the benchmark web-site RUBBoS [34](more details in Section 5.1), the capacity of each tier C_i can be calculated from the service time. We choose *heavy requests* (e.g., long service time by consuming more system bottleneck resource, detailed explanation in Section 4.2) as attack requests. Table 3 lists a group of reasonable values of the constants for our model profiled in RUBBoS. During the profiling, we choose 2000 legitimate users with 7-second think time as our baseline experiment. All the transactions supported by RUBBoS are terminated in MySQL, each transaction follows a static page-load terminated in Apache, and no transaction terminates in Tomcat. Thus, the request rate of each tier λ_i is 280, 0, and 280, respectively. We set the queue size of each server Q_i satisfying the condition *C1* in Equation (2).



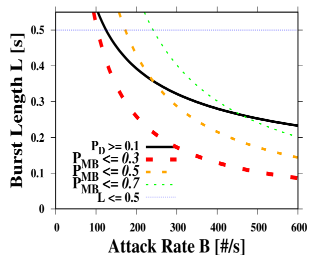
(a) There exists an overlapped feasible zone below the dash line P_{MB} and above the solid line P_D.



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



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



(d) Various target P_{MB}. No solution when P_{MB} is < 0.3s (red line), since no overlap exists.

Figure 2: Numerically solve the optimal attack parameters. Solid line depicts damage length, the target zone is above the solid line; and dash line depicts millibottleneck length, the target zone is below the dash line.

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

Table 3: Constant parameters estimation profiled in RUBBoS experiment with 2000 concurrent users.

Attack Goal and Solver. Suppose that we set our attack goal as 95th percentile response time longer than 1 second which is a severe long-tail latency problem for most e-commerce websites [6, 10, 11, 18], and the duration of a millibottleneck less than 0.5 seconds in the bottleneck tier such that the average utilization can be at moderate level (e.g., 50-60%) to bypass the defense mechanisms. If we assume the burst interval T is 2 seconds, then the input to the model can be two inequations: damage length P_D is bigger than 0.1 seconds and millibottleneck length P_{MB} is less than 0.5 seconds. Further, we turn these two inequations to L as a function of B, the others parameters are all constants(Inequation (8) and (9)).

$$L >= \sum_{i=1}^{n} l_i + 0.1 = \frac{Q_n}{(\lambda_n + B - C_{n,A})} + \frac{(Q_{n-1} - Q_n)}{(\lambda_{n-1} + \lambda_n + B - C_{n,A})} + \ldots + \frac{(Q_1 - Q_2)}{(\sum_{i=1}^{n} \lambda_i + B - C_{n,A})} + 0.1 \tag{8}$$

$$L <= 0.5 * (1 - \lambda_n * \frac{1}{C_{n,L}}) * \frac{C_{n,A}}{B} \tag{9}$$

Thus, the problem of selecting a set of optimal attack parameters (B,L,V,T) can become a nonlinear optimization problem. Although nonlinear optimization problem is hard to solve since there exist multiple feasible regions and multiple locally optimal points in those regions, we can add more constraints to narrow the range of feasible regions. For instance, the burst length obviously should be less than target millibottleneck length(e.g.,L <= 0.5).

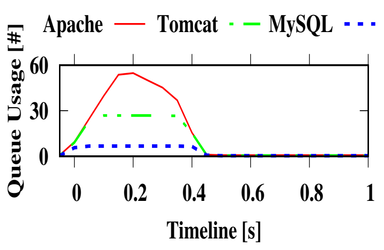
Substituting the constant parameters in Inequation (8) and (9), we can get an unique feasible region as the potential attack parameters shown in Figure 2a. In real cloud production settings, the queue size must be diverse according to the capacity of the websites. In Figure 2b, we can see that as the queue of the front tier (e.g., Apache) increases from 30 to 80, the feasible region reduces; when it increases to 130 (the red line), the two inequations do not overlap, which implies that there is no solution to satisfy our predefined attack goal. The fundamental reason is that our attack goal is too strict, which seems to be an impossible mission. Note that when the queue of the front tier is 80, in the strictest attack target cases (P_D >= 0.2 seconds, the red line in Figure 2c; or P_{MB} <= 0.3 seconds, the red line in Figure 2d), there is also no solution that can solve the attack parameters of our attacks. We will further discuss how to deal with the non-solution cases in Section 4.1.

3.3 Simulation Experiments

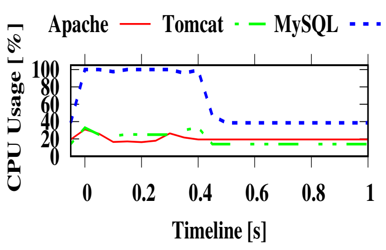
The numerical solver in the previous section does not consider many aspects of the real system (e.g., the competition of the free position in the queue between attack requests and normal requests, overloaded attack requests can be dropped, etc.). To further validate the simple model, we present results from Java Model Tools(JMT) [7] in which such limitations are absent. JMT is an open source suite for modeling Queuing Network computer systems. It is widely used in the research area of performance evaluation, capacity planning in n-tier systems. Thus, it is a natural choice to evaluate the impact of our attacks in n-tier systems. We modify the JMT code and simulate the bursts of attack requests for our attacks with the configurable attack parameters in our model.

Given the proposed model and the idea of solving the nonlinear optimization problem, we can get the feasible region of attack parameters. We initialize the parameters in JMT similar to the setting of our numerical solver, and choose a potential optimal point (400,0.215) in Figure 2a as our attack parameters, the attack rate B is 400 requests per second and the burst length L is 0.215 seconds, so the attack volume per burst V is 86 (see equation (1)) if all the attack requests will not be dropped by the target system.

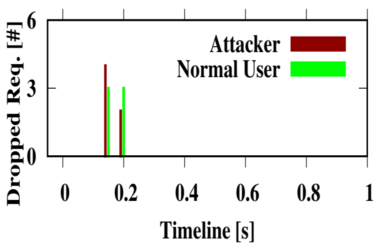
Results in JMT Figure 3 shows the results of one burst during 1 second time period using fine-grained monitoring (e.g., 50 milliseconds) in JMT expriment. Figure 3a illustrates the process of filling up all the queues in the n-tier system. Note that the queue of MySQL, Tomcat, and Apache is overflow from down-stream tiers



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



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



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

Figure 3: Results of one burst during 1 second period in JMT experiment

to up-stream tiers overtime. The CPU saturations of the bottleneck tier MySQL last approximately 400 milliseconds as shown in Figure 3b, less than the expected value 500 milliseconds calculated by our model, since overloaded attack requests are also dropped by the front tier which will not go through the front tier and into the bottleneck tier. Figure 3c shows dropped requests perceived by attackers and legitimate users in the corresponding burst. Note that the dropped requests span two sampling duration (100 milliseconds), validating our model expectation for the dropped length. The interesting observation is that the dropped requests from attackers is bigger than ones from legitimate users at the time of 0.15, the opposite phenomenon happens in the next sampling windows at the time of 0.2. This implies that the requests from the attacker and ones from the legitimate users compete the available position of the queue freed by the outgoing request in the n-tier system during *damage period* [28], eventually the loser will be dropped.

However, when we aggregate the data of the legitimate users during the 3-minute simulation experiment, the amount of dropped requests is 1099 and the total requests is 54901, thus the actual drop ratio for the legitimate users is 2%, which is far from the predefined goal of the drop ratio 5%. From this observation, we should calibrate the drop ratio from Equation 6 to

$$\rho(\mathrm{T}) = \frac{P_D * \rho(\mathrm{L})}{T} \tag{10}$$

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

Tuning the Attack Parameters. Due to the existence of the competition between the attackers and the legitimate users, the attackers may fail to get the predefined attack goal (e.g., 95th percentile response time > 1 second) by using the recommended attack parameters of the proposed model. We further investigate how to increase the competition ability of attack requests, and the drop ratio of the requests from the legitimates users during damage length, namely ρ(L). Finally, the attackers can choose the optimal attack parameters to achieve high damage with low cost and high stealthiness. For simplicity, we assign attack interval T as fixed value (say, 2 seconds), since our focus on this paper is to investigate how to effectively trigger the millibottlenecks which is predominantly determined by

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

Table 4: Fix burst volume V=86. Bigger B Higher $\rho(T)$, implying higher competition ability for the burst with larger B.

the other three parameters (B,L,V). Due to interdependent relationship of these three parameters in Equation 1, we fix one parameter (L or V), then observe the impact with various attack rate B. We still consider the marked potential optimal point (400,0.215) in Figure 2a as our baseline attack parameters.

First, we fix burst volume V as 86, and select a set of attack rate (from 300 to 800) to conduct our attack experiments using JMT. Table 4 shows that as the attack rate increases, the drop rate $\rho(T)$ accordingly increases, which confirms that higher attack request rate, compared to normal request rate, can achieve higher competition ability to seize the available position in the queue.

Next, we fix burst length L as 0.215 seconds, and select another set of attack rate to conduct our JMT experiments. Figure 4 depicts $\rho(T)$ and $\rho(L)$ as a function of the ratio of attack rate and service rate for the bottleneck tier. We mark two vertical lines to split up into three zones with various attack rate B. (1) In zone a, B is less than $C_{n,A}$, the drop ratios are all zero, since the attack rate is too low to trigger an effective millibottleneck to lead to cross tier queue overflown in the target system, which violates the condition $C2$ in Equation 2. (2) In zone b, B is bigger than $C_{n,A}$, the drop ratio increases non-linearly as B increases. Observe that ρ(L) of Normal Users is a little bit bigger than ρ(L) of Attackers, because B is bigger than λ_n. In this case, the requests from the attackers can seize the available position in the queue with a more highly probability than ones from the legitimate users. (3) In zone c, B is bigger than $C_{1,A}$, the attack requests are directly dropped by the most front tier, thus the increase of B does not contribute to the drop ratio of

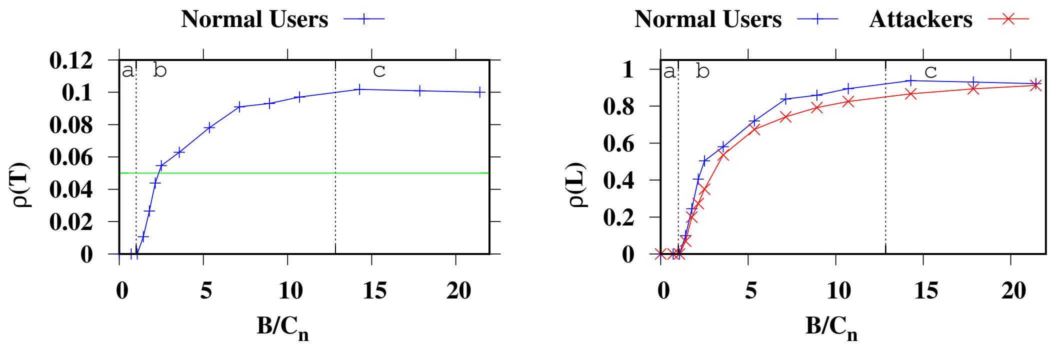


Figure 4: Fix burst length *L*=0.215 seconds. (1) in Zone a, $B < Cn$, *B* is too low to trigger effective millibottlenecks; (2) in Zone b, $B > Cn$, the bottleneck is in *n*-th tier, $\rho(T)$ increases as *B* increases; (3) in Zone c, $B > C1$, the attack requests are directly dropped by the most front tier, no obvious attack effect ($\rho(T)$ is flat) even though *B* increases tremendously, implying that we should choose a moderate *B* until the attack goal is achieved (e.g., the green horizontal line is a target).

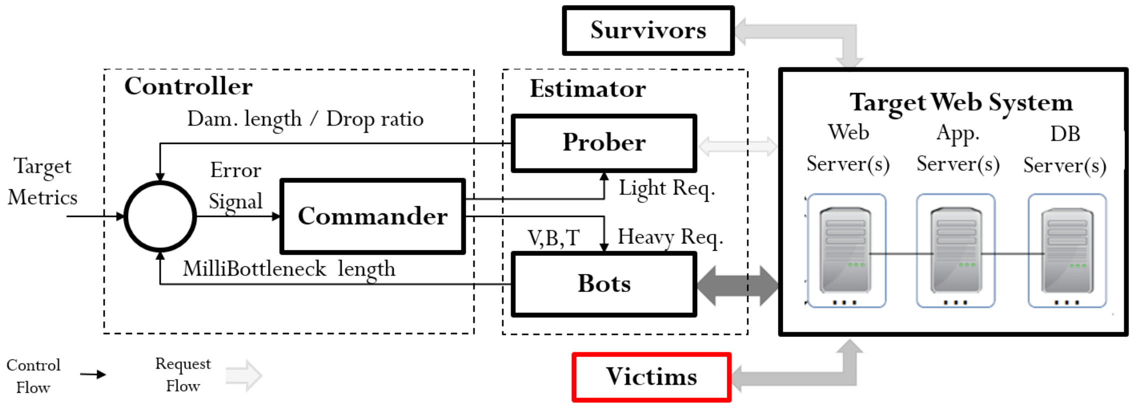


Figure 5: A feedback control framework

the legitimate users, it only increases the drop ratio itself. Given this observation, we should choose a moderate *B* until the attack goal is achieved (e.g., the green horizontal line in Figure 4).

4 TAIL ATTACKS IMPLEMENTATION

4.1 Overview

As mentioned before, the analytical model used in the numerical solver analyzes the simple scenario skipping many aspects of the system reality (e.g., the competition for the free position of the queue between attack requests and normal requests, overloaded attack requests can be dropped, etc.), and the simulation experiments show more complicated cases involving the absent system reality. However, our attacks do not consider more realistic case with dynamics of baseline workload [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 Tail Attacks may become failed ones over time, either it can not trigger millibottlenecks (not enough attack requests) or it might trigger the defense alarm of the target system (too frequent or massive attack requests). How can we dynamically adjust the attack parameters catering to instant system state and baseline workload is a big challenge for Tail Attacks. The static attack parameters in the dynamical environment may make the attack either invalid like a “mosquito bite" or easily exposed to the detection mechanisms. In this section, we implement a feedback control-theoretic (e.g., Kalman filter) framework that allows attackers to fit the dynamics of background requests or system state by dynamically adjusting the optimal attack parameters in Figure 5.

Via our best practice, we find that the attack rate should not be invariable to maximize the attack effectiveness and stealthiness. [25] suggests a double-rate DoS steam to minimize the attack cost. We design a three-stage transmitting strategy to send one burst: Quickly-Start , Steadily-Hold and Covertly-Retreat. In Quickly-Start stage, the attacker sends the burst of requests at a high rate to quickly fill up all the queues in the n-tier system, *heavy requests* (detailed explanation in Section 4.2) are preferred because it can consume more bottleneck resources and occupy the queue longer with low cost and high stealthiness, the amount of heavy

[1]For e-commerce applications, the baseline workload during the day time is usually significantly higher than that during the mid-night period.

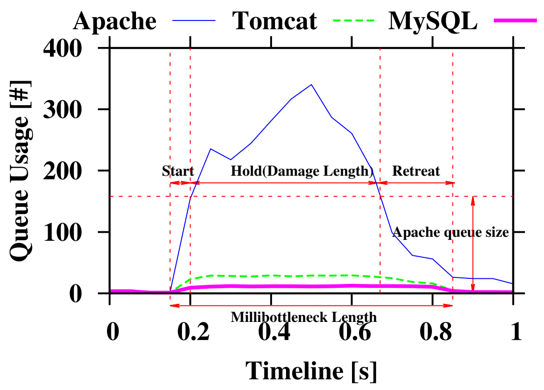


Figure 6: Queue shifts during a three-stage burst(quickly-start, steadily-hold, covertly-retreat)

requests during this stage should be large enough to temporarily saturate the bottleneck resource in the target system [40]. In Steadily-Hold Stage, the attacker should guarantee the queue can be overflown during this stage and attack requests can seize the free position in the queue with highly probability. *Heavy requests* are not necessary to serve as attack requests during this stage. We prefer *light requests* to hold on queue, such that in the last Covertly-Retreat stage, the attack requests can quickly and covertly leave the systems. In Covertly-Retreat stage, there is no attack request to be sent out. Figure 6 demonstrates the queue shifts during a three-stage burst. Through the strategy of variable attack rate and various attack requests, we can solve the insolvable cases in Section 3.2 by carefully choosing less heavy requests as attack requests.

Return back to our model in Equation 5, we can see the relationship between P_D and the attack rate B were nonlinear. We can transfer P_D as a function of V and B using equation 1:

$$P_D = \frac{V}{B} - \sum_{i=1}^{n} l_i \tag{11}$$

If we fix the attack rate B, mathematically, P_D and the attack volume V have a linear relationship; the same as P_{MB} (see Equation 7). These linear relationships provide us with a firm theoretical foundation to dynamically adapt the optimal attack parameters fitting the changes of system state and baseline workload. The overall control algorithm can be described in Algorithm 1.

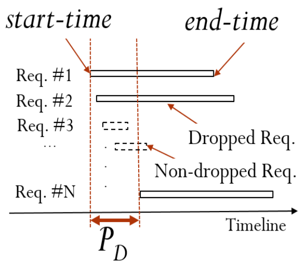
Algorithm 1 Pseudo-code for the control algorithm

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

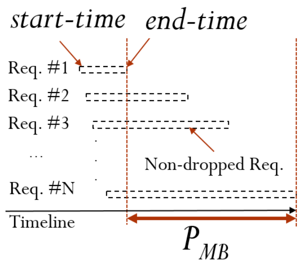
4.2 Estimator

The Estimator, as illustrated in Figure 5, estimates three critical metrics in the control algorithm implementing the proposed model: service time of the requests, damage length P_D, and millibottleneck length P_{MB}. We use a prober to monitor attacks and infer P_D, coordinate and synchronize bots to launch attacks and infer P_{MB}.
Estimating Service Time. Service time of a HTTP request is the time that the target web system needs to process the request without any queuing delay. It is easy to calculate the end-to-end response time of a request using two time stamp of sending requests and receiving responses [26], we term them *start-time* and *end-time* of a HTTP request. The end-to-end response time of a request equals the difference between *end-time* and *start-time*. Typically, the end-to-end response time of a HTTP request involves three parts: the network latency between the client and the target web application, the queuing delay in the n-tier system, and the service time of each server. The network latency can be measured using the *ping* command. When the target system is at low utilization, the queuing effect inside the target system can be ignored. Thus, we can approximately estimate the service time of any HTTP request supported by the target web system as the end-to-end response time subtracting the network latency when the target system is in the time block with a low workload. Since the service time of the estimated request may drift over time (e.g., due to changes in the data selectivity and the network latency variation) in real applications, we measure the service time of a HTTP request multiple times and take the average.

Previous research results [41] show that the predominant part of the service time of a request is spent on the bottleneck resource in the system. We call the requests that heavily consume the bottleneck resource as *heavy requests* with long service time (e.g., the request querying multi-tables in the database) while those consume no or little bottleneck resource as *light requests* with short service time (e.g., static requests) [40]. Thus, the prober naturally exploits *light requests* to monitor the impact of the attacks since it can be more elusive under the radar without causing any alert of the target web



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



(b) Infer P_{MB} by *end-time* of the last non-dropped attack request subtracting *end-time* of the 1st non-dropped attack request during a interval.

Figure 7: Demonstration of inferring damage length and millibottleneck length by Estimator.

system; and the bots can take *heavy requests* as candidate attack requests since it can be more efficient causing millibottlenecks and cross tier queue overflow. More advance technology about profiling *heavy requests* will be discussed in Section 7.

Through profiling and exploiting heavy requests [40], Tail Attacks can transiently saturating the critical bottleneck resource (e.g., database CPU) of the target systems, which can saturate the critical resource of the system with much lower volume (thus less bots are needed) compared to that of traditional flooding DDoS attacks which usually try to fully saturate the target network bandwidth.

Estimating *damage length* P_D. To estimate P_D, the prober needs to send probing requests (e.g., *light requests*) to the target web system at a predefined rate and record *start-time* and *end-time* of a probing request. The recommended sending interval of probing requests is less than the target damage length, in the case, any probing request may not miss the period of overflown queue caused by an attack burst and the prober can sense P_D [22]. Figure 7a illustrates the implementation approach to estimate P_D as *start-time* of the last dropped probing request subtracting *start-time* of the first dropped probing request during a burst. Since some probing requests may probably seize the free position in the queue and are not be dropped during the damage length, we can calibrate P_D by multiplying ρ(L) (see equation 10). Some websites may send some alarm to the users if they send the requests at a very high rate. In this case, the prober can send the probing requests at an acceptable rate for the target web system and estimate the drop ratio during a sampling period, then our control algorithm can exploit drop ratio as the target criterion to dynamically adjust the attack parameters.

Estimating *millibottleneck length* P_{MB}. After sending a burst of attack requests (e.g., *heavy requests*) to the target web system, the bots can record *start-time* and *end-time* of an attack request and estimate P_{MB}. We only count the non-dropped attack requests, since the dropped request involves TCP retransmission time-out. There are two options to infer P_{MB}. One way is *end-time* of the last non-dropped attack request subtracting *start-time* of the first non-dropped attack request during one attack interval. As we mention before, the end-to-end response time of a HTTP request involves three parts: the network latency, the queuing delay, and the service time. In this way, the result overcharges a length of the network latency. The other option of inferring P_{MB} is *end-time* of the last non-dropped attack request subtracting *end-time* of the first non-dropped attack request during one attack interval shown in Figure 7b. In this way, it undercharges a length of the service time. For the concrete environments of a special website, the attacker can choose any approach to estimate millibottleneck length. In our evaluation section, we choose the second one, since the network latency is much longer than the service time in our RUBBoS environment.

4.3 Controller

So far, we discuss many aspects that can influence the effectiveness of our attacks. In the model, the competition for the free slot of the queue between attack requests and normal requests may impact precision of damage length, and the dropped attack requests may decrease millibottleneck length. For Estimator, the network latency variation and the drifted target system state might reduce the accuracy of inferring damage length and millibottleneck length in our implementation. All of aspects lead to the observing and measuring inaccuracy, and result in the invalidation of launching an effective attacks using our control algorithm. To mitigate these negative impacts for our control algorithm, we adapt a popular feedback-based control tool, Kalman filter [20]. On the one hand, it can take past measurements into account for implementing our feedback control algorithm, reducing the impact of the process noise (e.g., baseline workload and system state). On the other hand, it can mitigate the measurement noise due to inaccuracy of the estimator.

Let *z(k)* be the measurement of P_D in k-th burst by Estimator. Since P_D is a linear function of burst volume V (see equ. (11)), we can define *x(k)* using a linear dynamical system model:

$$SystemDynamics : x(k) = x(k-1) + U(k) + v(k) \tag{12}$$

$$MeasurementDomain : z(k) = x(k) + w(k) \tag{13}$$

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

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

Predict (Time Update)

$$\hat{x}(k \mid k-1) = \hat{x}(k-1 \mid k-1) + U(k) \tag{14}$$

$$\mathrm{P}(k \mid k-1) = \mathrm{P}(k-1 \mid k-1) + V(k) \tag{15}$$

Correct (Measurement Update)

$$\mathrm{K}g(\mathrm{k}) = \frac{\mathrm{P}(k \mid k-1)}{(\mathrm{P}(k \mid k-1) + W(k))} \tag{16}$$

$$\hat{x}(k \mid k) = \hat{x}(k \mid k-1) + Kg(k)(z(k) - \hat{x}(k \mid k-1)) \tag{17}$$

$$P(k \mid k) = (1 - Kg(k))P(k \mid k-1) \tag{18}$$

where *W(k)* and *V(k)* are the covariances of w(k) and v(k), respectively. In practice, we can estimate these two noise covariances using automatic mathematical tools (e.g., autocovariance least-squares method [32]) or manual observation to tune the optimal value. *Kg(k)* is termed the Kalman gain which represents the confidence index of the new measurement (*z(k)*) over the current estimate ($\hat{x}(k \mid k-1)$). If *Kg(k)* equals 1, it implies that the attacker totally trust the measurement, the effectiveness of adjusting the attack parameters in our attacks is totally depending on the accuracy of the estimated value by Estimator.

Using the Kalman filter, the Controller in Figure 5 can predict the required attack parameters at *k-th* burst given the historical results of all *k-1* bursts, dynamically command the new parameters to the bots, and automatically and effectively launch Tail Attacks.

5 REAL CLOUD PRODUCTION EVALUATION

5.1 Tail Attacks in Real Production Settings

To evaluate the practicality of our feedback control attack framework in the real cloud production settings, we deploy a representative benchmark website in the most popular two commercial cloud platforms (Amazon EC2, Microsoft Azure) and one academic cloud platform (Cloudlab[31]).

Experiment Methodology. We adopt RUBBoS [34], a representative n-tier web application benchmark modeled after the popular news website *Slashdot.* We configure RUBBoS using the typical 3-tier or 4-tier architecture. A sample setting *EC2-1414-6K* in Table 5 is 1 Apache web server, 4 Tomcat application servers, 1 C-JDBC clustering load-balance server, 4 MySQL database servers deployed in Amazon EC2, and 6000 concurrent legitimate users. RUBBoS has a workload generator to emulate the behavior of legitimate users to interact with the target benchmark website. Each user follows a Markov chain model to navigate among different webpages, with averagely 7-second think time between every two consecutive requests. Through modifying the RUBBoS source code, we simulate various baseline workload (e.g., variable concurrent users during the experiment). Meanwhile, in our experiments we adopt a centralized strategy to coordinate and synchronize bots [12, 37, 49] (more discussion about distrubuted bots coordination and synchronization in Section 7). All the bots are in the same location to rule out the impact of the shift of network-latency. We control a small bot farm of 10 machines (one of which serves as a centralized controller), synchronized by NTP services, which can achieve millisecond precision [15]. Each bot uses *Apache Bench* to send intermittent bursts of attack HTTP requests, commanded by our control framework.

All the VMs we run are 1 vCPU core and 2GB memory, which is the basic computing unit for the commercial cloud providers. We select HDD disk since our experimental workloads are browse-only CPU intensive transactions. We select t2.small instance ($0.023 per hour) in Amazon EC2 us-east-1a zone, and A1 ($0.024 per hour) instance in Microsoft Azure East US zone. They have similar

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

W/o att.: Without attacks, **Att.**: under Tail Attacks,
CPU: MySQL CPU usage, **NW**: Apache network traffic

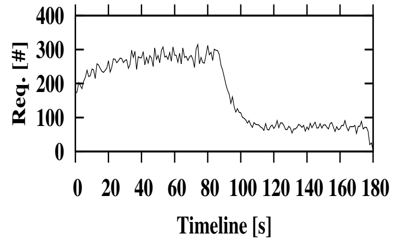
Table 5: The corresponding parameters and bottleneck resource utilization of Tail Attacks in real production settings.

prices and hardware configurations. However, the CPU core in EC2 (2.40GHz Intel Xeon E5-2676 v3) is more powerful than the one in Azure (2.10GHz AMD Opteron 4171 HE or 2.40GHz Intel Xeon E5-2673 v3). The worst one is in NSF Cloud (2.10GHz Intel Xeon E5-2450), where we run the VMs in Apt Cluster in the University of Utah's Downtown Data Center .

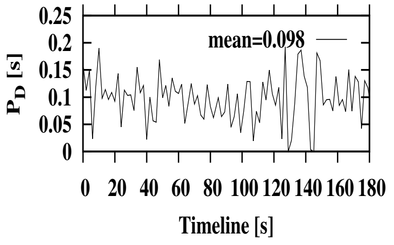
For the baseline workload, we have two chosen criteria: the bottleneck resource utilization (e.g., CPU utilization or Network bandwidth) is less than %50 (Column 6 and 8 of Table 5), and no long-tail latency problem exists in without-attacks cases shown in the Table 1. Because EC2 has more powerful CPU that we used in our experiments, it can serve higher baseline workload than the other two. The network overhead can be the bottleneck resource in EC2 platform even though CPU is at low utilizations [17]. In our experiments, MySQL CPU is the bottleneck tier in Azure and NSF Cloud due to the high resource consumption of database operations.

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

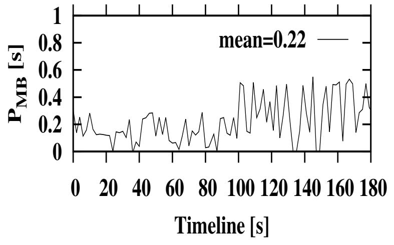
Results. Column 2 to 5 of Table 5 show the corresponding model parameters in our real cloud production setting experiments controlled by our attack framework. It clearly shows that our attacks controlled by our algorithm can achieve the predefined targets (5% drop ratio ρ(T), damage length P_D < 100ms, millibottleneck length P_{MB} < 500ms). EC2 has more powerful CPU, so it requires more attack requests per burst V to launch a successful attack. As the servers scale out (from 111 to 1212, 1414), the n-tier system can service more legitimate users, at the same time, in order to launch Tail Attacks it requires higher V due to their higher capacity, which means that bigger websites need larger botnet to attack. Column 7



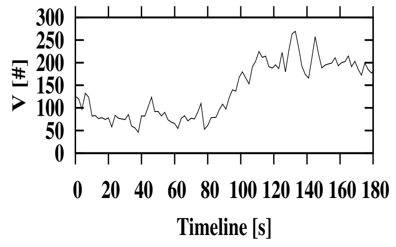
(a) The requests of baseline workload vary from 2000 to 500 concurrent users.



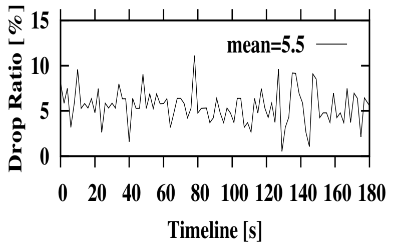
(b) Damage length inferred by the prober nearly equals the target 100ms.



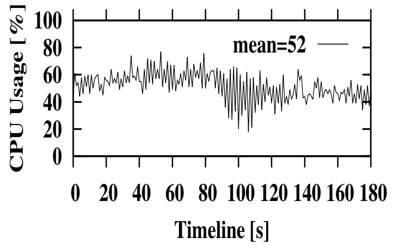
(c) Millibottleneck length estimated by the bots is less than 500ms, achieving the goal.



(d) Required attack volume V controlled by our framework increases as background requests decreases in Fig. 8a.



(e) Drop ratio for legitimate users meets the target 5%, indicating the effectiveness controlled via damage length P_D in Fig. 8b.



(f) Bottleneck resource utilization approximately equals 50%, indicating the effectiveness controlled via P_{MB} in Fig. 8c.

Figure 8: Results of Tail Attacks on RUBBoS web application in a real cloud production setting

and 9 of Table 5 show the average CPU utilization of MySQL tier and the average network traffic of Apache tier, which are the bottleneck resource in our experimental environment. Under our attacks in the real cloud production settings, the end users encounter the long-tail latency problem (see Table 1). However, all the bottleneck resources are under moderate average resource utilization even in large scale case (e.g., CPU is 25.6% in Azure-1414-3K, and NW is 401MB/s in EC2-1414-6K). These experimental results show that our attacks can cause significant long tail latency problem, while the servers are far from saturation, guaranteeing a high level stealthiness.

Figure 8 further illustrates the 3-minute detailed experimental results with various baseline workload under our attacks launched by our control framework in NSF-1212 setting. Figure 8a shows baseline workload changes from 2000 to 500 concurrent users at the middle point of the experiment (each user has 7-second think time between two webpages). Note that the shift of baseline workload in this case is different from the previous cases in which we scale out the servers. Figure 8d shows the required burst volume dynamically adjusted by our framework, a sudden increase at the middle due to the decrease of baseline workload. Figure 8b and Figure 8c depict damage length and millibottleneck length estimated by the prober and the bots, respectively. The measured average value of two critical metrics of our model are successfully controlled in the target range. Figure 8e and Figure 8f depict the drop ratio during every attack interval and the CPU utilization of MySQL using 1 second granularity monitoring, respectively, which match our predefined attack goal to a great extent. In general, through the RUBBoS experiments in real cloud production settings, we can validate and confirm the practicality of our attack control framework.

5.2 Tail Attacks under IDS/IPS systems

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

Experiment Methodology. Typically, the popular solution to mitigate the application layer DDoS attacks is identifying abnormal user-behavior [38, 44–47]. Snort [9] is a signature rules based Open-Source IDS/IPS tool that is widely used in practice for DDoS defense, the users can customize the alert rules by setting reasonable thresholds in the specific systems. We set alert rules following some user-behavior models in Snort to evaluate whether our attacks deviate from the model (judging based on predefined thresholds). In the following experiments, we configure 2000 concurrent legitimate users, and our attack goal is to achieve the 95th percentile response time longer than 1s for these legitimate users.

Results. To validate the user behavior model, we take request dynamics model [33] as an example. The authors analyzed the distribution of a user's interaction with a Web server from a collection of Web server logs, categorized four session types (searching, browsing, relaxed and long session), and modeled the features for each type of session based on average pause between sessions. Typically, average inter-request interval for searching and browsing sessions is less than 10 seconds. RUBBoS [34] also models the inter-request interval per session as a Poisson process with the mean as 7, which means an average 7-second think time between two webpages for each session. In this case, we can calculate the 95% confidence interval as (2.814, 14.423). We can set the minimal boundary of the interval (e.g., rounded to 3, the statistical granularity is seconds in

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

Table 6: The attacker can successfully predict the inter-request model. Less the sampling period, higher alerts from the legitimate users, indicating higher false positive error. However, the attacker can trigger no alert by carefully controlling the request pattern.

Snort) as the alert threshold to validate whether our attacks deviate the RUBBoS model, since the smaller alert threshold can lead to less false positive error. We use the "detection_filter" property in Snort to define the alert rules monitoring inter-request rate for each IP, which generate an alert once the amount of requests from the same IP exceeds the predefined threshold during a sampling statistical period. Column 1 and 2 of Table 6 depict the threshold and the sampling interval for these alert rules, respectively.

We use RUBBoS client to simulate 2000 concurrent legitimate users and craftily control our bots catering to the inter-request model in RUBBoS to bypass the above alert rules in Table 6 while achieving our attack goal. Due to the limited numbers of IPs, we can not have enough IP address to simulate the 2000 legitimate users and the bots from real IP address to evaluate the above detection rules in Snort. However, we can map a session to a individual IP address and implement the same detection algorithm using the "detection_filter" property of Snort to validate the above alert rules. We conduct a 3-minute successful attack experiment in our RUBBoS websites. In our case, to achieve the attack goal, the required attack parameters are V as 300, L as 50 and I as 3. To follow the inter-request model in RUBBoS (alert threshold as 3), it requires 301 totally-synchronized bots (one session simulates one bot in our experiments and sends one request in more than 3s interval) while avoid triggering the alerts (deviating from the model). Column 3 and 4 of Table 6 report the traced alert number from the bots and the legitimate users for these rules, respectively. As a result, our attacks can be totally invisible to these alert rules. Note that as the sampling interval increases the alerts from the legitimate users decrease, the reasonable sampling interval can reduce the false positive errors (typically the interval should be on the order of minutes). Another important guide to our attacks is that as the sampling interval increases, the threshold of the rules also has to increase, which can give us more flexible options to send the attack requests using different intervals (e.g., in the above case, less than 10 requests every 30s interval per session, or less than 20 requests every 60s interval per session). To choose which sending pattern, we can further learn from the other user-behavior models to make our attacks even more stealthy.

Someone may argue that the "no alert" results are got from the assumption that the attackers comprehensively know the alert

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

1G.: 1Group, **2G.**:2Groups, **4G.**:4Groups, **8G.**:8Groups
minS.:minimal Size, **minG.**:minimal Group

Table 7: The attacker conservatively predicts the inter-request model. To achieve a burst with 300 requests in every 3 seconds (V=300, L=50, I=3), more conservative predict needs bigger botnet.

thresholds of the user-behavior model, such that they can design a corresponding attack pattern to avoid be detected. Most user-behavior models are public to both the defenders and the attackers, the only gap is the specific alert threshold and rules, which typically are learned from the server's past logs for their anomaly detection systems by the defenders of specific websites [33]. However, the attackers can use the questionnaire approach to similarly estimate the real users' behaviors, and use a conservative value as the potential threshold to design the attack pattern with the price of increasing more bots shown in Table 7. The four rows in Table 7 show four cases with different predicted values (3, 6, 12, 24). Obviously, as the predicted value is more conservative, it requires a bigger botnet (minimal size is 2400 when the prediction is 24). In the '24' case, the attacker must split the 2400 bots into 8 groups, each group takes turn to send a burst of 300 requests in every 24-second interval.

6 DETECTION AND DEFENSE

Here, we consider a solution to detect and defend against Tail Attacks. There exists no easy approach to accurately distinguish the attack requests from legitimate requests. Instead, we can identify the attack requests by detecting the burst and matching the bursty arrivals to millibottlenecks and cross tier queue overflown (Event2 and Event3 in Section 2). We present a workflow to mitigate our attacks involving three stages: fine-grained monitoring, burst detection, and bots blocking.

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 [41–43]. 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, 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 Tail Attacks, the next task is to distinguish the requests of the bots from the requests of the legitimate users during the burst and block them. The attacker, in our attack scenario, aims to coordinate and synchronize the bots to sending bursts of attack requests during short "ON" burst period and repeat the process after long "OFF" burst period as introduced in Section 2, we can ideally introduce a new request metric that quantifies the suspicion index of the incoming requests by aggregating the requests statistics during "ON" burst and "OFF" burst for further analysis. Specially, we define the *suspicion index* for each IP address as follows:

$$\mathrm{SI}_{IP} = \frac{\mathrm{NB}_{IP}}{\mathrm{N}_{IP}} \tag{19}$$

where NB_{IP} and N_{IP} are the number of requests for each IP during "ON" burst and the attack interval T (including "ON" and "OFF" burst), respectively. If SI_{IP} for a IP is close to 1, the IP is likely to be a bot; on the other hand, SI_{IP} of the legitimate user can be approximately the target attack drop ratio (e.g., 0.5 if the target is 95th percentile response time longer than 1 second). Figure 9 shows Probability Density Function of SI_{IP} in the RUBBoS experiment in Section 5.1. 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.

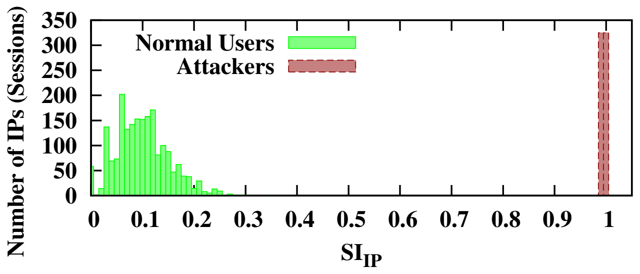


Figure 9: Identify bots

7 DISCUSSIONS

Impact of load balancing. Some web applications adopt load balancing (e.g., Amazon Elastic Load Balancing [4]) in front of the system to distribute load across multiple web servers. This type of load balancing works well for stateless servers such as web servers, in which incoming traffic are supposed to evenly distribute among them. However, whether or not load balancing can mitigate the effectiveness of Tail attacks depends on the location of the bottleneck resource in the target web system. For example, if the bottleneck resource is in the web server tier, load balancing indeed increases the bar of an effective Tail attack since each web server only receives a portion of total attacking requests, thus higher volume of attacking requests per burst are needed to create millibottlenecks in the system. On the other hand, if the bottleneck resource is in a non-scalable tier such as the relational database tier, the load balancing in front tiers does not help mitigating the effectiveness of Tail Attacks. This is because no matter which web server an attacking HTTP request arrives at, the database queries sent out by the HTTP request may eventually converge to the same database server, and create millibottlenecks there.

Impact of cloud scaling. Large-scale web applications usually adopt dynamic scaling strategy (e.g., Amazon Auto Scaling [2]) for better load balancing and resource efficiency, however, Tail Attacks can easily bypass current dynamic scaling techniques, since the control window of the state-of-the-art scaling mechanism is usually in minute-level (e.g., Amazon CloudWatch monitoring in a minute granularity), while Tail Attacks are too short (sub-second duration) for them to catch and take any scaling actions [40]. The main advantage of Tail attacks is that it is invisible to most monitoring programs and can remain hidden for a long time because of the low volume characteristic and the sub-second duration of millibottlenecks as shown in our experimental results.

Browser compatibility check. Some state-of-the-art defense tools may check the header information of each HTTP request to determine whether it is sent from a real browser or from a bot. A HTTP request sent from a real browser usually has completed header information such as "User-Agent", while such information may not appear or be difficult to be generated by a bot which only uses a script language to generate HTTP requests. In addition, some websites such as Facebook need a legitimate user to login first before any following transactions, especially *heavy query requests* (detailed explanations in Section 4.2). They may track the cookies stored in the client side browser in order to keep an active session in the server side; a bot may not be able to interact with such websites due to the lack of support of a real browser. We can address these challenges by using PhantomJS [36] to generate attacking HTTP requests. PhantomJS is a headless web browser without a heavy graphical user interface. It is built on top of WebKit, the engine behind Chrome and Safari. So PhantomJS can behave the same as a real browser does. Therefore, an attacker can launch browser-based Tail Attacks using heavy requests as attack requests by PhantomJS, and the generated requests will be extremely difficult to distinguish from the requests sent by legitimate users.

Distributed bots coordination and synchronization. One precondition of Tail attacks is that bots could be coordinated and synchronized so that the generated HTTP requests are able to reach the target web system within a short time window. Many previous research efforts already provide solutions, using either centralized [12, 37, 49] or decentralized methods [22], to coordinate bots to send synchronized traffic to cause network congestion at a specific network link. Centralized control can achieve higher level of bots coordination and synchronization, which enables a more effective Tail Attack compared to decentralized methods. In this paper we adopt the centralized control method to do experiments. On the other hand, a decentralized method in general is able to coordinate and synchronize more bots than a centralized one, thus making it possible to target large-scale/high-capacity websites.

However, a decentralized method is more challenging to control the length of each burst of attacking requests arrived at the target website, thus mitigating the effectiveness of Tail Attacks.

8 RELATED WORK

In this section, we review the most relevant work with regard to low-volume application-layer attacks, which is even stealthier to avoid traditional network-layer based detection mechanisms [30, 35, 48].

Low-volume Application Layer DDoS attacks. Low-volume DDoS application attacks are characterized by a small number of attack requests transmitted strategically to the target application servers, as an extension of network-layer low-volume attacks [12, 14, 21, 22, 25, 27, 39]. Macia-Fernandez et al. initially proposed *low-rate* attacks against application servers (*LoRDAS*) [28] that send traffic in periodic short-time pulses at a low rate, sharing the similar on-off attack pattern with our attacks. *Slow-rate* attacks [8] deplete system resources on the server‘s side by sending (e.g., slow send/Slowloris [13]) or receiving (e.g., slow read) traffic at a slow rate. Our attacks share the similar features of low attack volume with low-rate and slow-rate attacks.

However, compared to these two attacks, our attacks dig more deeply into n-tier architecture applications while *LoRDAS* attacks only in 1-tier application server. Our analytical model for n-tier systems can guide and guarantee our attacks dynamically controlled in a more effective and elusive way by accurately estimating *damage length* and *millibottleneck length*. In addition, slow-rate attacks need to develop well-crafted HTTP header (Slow Headers) or body (Slow Body) thus expose obvious attack patterns to the defense tools, while Tail Attacks use the legitimate and normal heavy requests as our attack requests thus hide deeper. More importantly, we exploit the ubiquity of millibottlenecks (with sub-second duration) and strong dependencies among distributed nodes for web applications, leading to long-tail latency problem with a higher level of stealthiness than *LoRDAS* and *Slow-rate* attacks.

Detecting and Defending against Application DDoS attacks. To mitigate application DDoS attacks, existing solutions typically focus on distinguishing application-layer DDoS traffic from the traffic created by legitimate users, such as abnormal user-behavior in high-level features [44, 46, 47] of surging web pages, in session-level [38], or in request-level [45]. To be stealthy, for the features of navigating web pages, our attacks can learn from the user behaviors of a legitimate user; as for session or requests level of our attacks, we can calculate the required optimized attack volume and botnet size discuss in Section 5. Compared to existing solutions, our proposed countermeasure can be tied to the unique feature of our attacks and accurately capture the bots.

9 CONCLUSION

We described Tail Attacks, a new type of low-volume application layer DDoS attack in which an attacker exploits a newly identified system vulnerability (millibottlenecks and resource contention with dependencies among distributed nodes) of n-tier web applications to cause the long-tail latency problem of the target web application with a higher level of stealthiness. To thoroughly comprehend the attack scenario (Section 2), we formulated the impact of our attacks in n-tier systems based-on queueing network model, which can effectively guide our attacks in a stealthy way (Section 3). We implemented a feedback control-theoretic (e.g., Kalman filter) framework that allows attackers to fit the dynamics of background requests or system state by dynamically adjusting the optimal attack parameters (Section 4). To validate the practicality of our attacks, we evaluated our attacks through not only analytical, numerical, and simulation results but also benchmark web applications equipped with state-of-the-art DDoS defense tools in real cloud production settings (Section 3 and Section 5). We further proposed a solution to detect and defense the proposed attacks, involving three stages: fine-grained monitoring, identifying bursts, and blocking bots (Section 6). More generally, our work is an important contribution towards a comprehensive understanding of emerging low-volume application DDoS attacks.

A DERIVATION OF *MILLIBOTTLENECK LENGTH*

During *millibottleneck length*, the bottleneck resources sustain saturation. *Millibottleneck length* should involve the serving time for both attack and normal requests during a burst. The attackers only send requests within the short "ON" period, thus the amount of attack requests is the burst volume *V*. Meanwhile, the legitimate users always send requests during *millibottleneck length*, thus the saturation length is an infinite recursive process until it converges to zero exponentially. Equation (20) represents *millibottleneck length* derived through the geometric progression in mathematics, here, *m* limits to infinity.

$$\begin{aligned}
P_{MB} &= V * \frac{1}{C_{n,A}} + V * \frac{1}{C_{n,A}} * \lambda_n * \frac{1}{C_{n,L}} \\
&\quad + V * \frac{1}{C_{n,A}} * \lambda_n * \frac{1}{C_{n,L}} * \lambda_n * \frac{1}{C_{n,L}} + \dots \\
&\quad + V * \frac{1}{C_{n,A}} * (\lambda_n * \frac{1}{C_{n,L}})^m \\
&= \lim_{m\to\infty} \sum_{k=0}^{m} V * \frac{1}{C_{n,A}} * (\lambda_n * \frac{1}{C_{n,L}})^k \qquad (20) \\
&= V * \frac{1}{C_{n,A}} * \lim_{m\to\infty} \frac{(1 - (\lambda_n * \frac{1}{C_{n,L}})^{m+1})}{(1 - (\lambda_n * \frac{1}{C_{n,L}}))} \\
&= V * \frac{1}{C_{n,A}} * \frac{1}{(1 - (\lambda_n * \frac{1}{C_{n,L}}))}
\end{aligned}$$

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

ACKNOWLEDGMENTS

This research has been partially funded by National Science Foundation by CISE's CNS (1566443, 1421561), SAVI/RCN (1402266, 1550379), CRISP (1541074), SaTC (1564097) programs, an REU supplement (1545173), Louisiana Board of Regents under grant LEQSF (2015-18)-RD-A-11, and gifts, grants, or contracts from Fujitsu, HP, Intel, and Georgia Tech Foundation through the John P. Imlay, Jr.

Chair endowment. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funding agencies and companies mentioned above.

REFERENCES

[1] Akamai. 2016. 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. (2016).

[2] Amazon. 2017. Amazon Auto Scaling. https://aws.amazon.com/documentation/autoscaling. (2017).

[3] Amazon. 2017. Amazon EC2. https://aws.amazon.com/ec2/. (2017).

[4] Amazon. 2017. Amazon Elastic Load Balancing. https://aws.amazon.com/elasticloadbalancing/. (2017).

[5] Chris Baraniuk. 2016. DDoS: Website-crippling cyber-attacks to rise in 2016. http://www.bbc.com/news/technology-35376327/. (2016).

[6] Salman A Baset. 2012. Cloud SLAs: present and future. *ACM SIGOPS Operating Systems Review* 46, 2 (2012), 57–66.

[7] Marco Bertoli, Giuliano Casale, and Giuseppe Serazzri. 2006. Java modelling tools: an open source suite for queueing network modelling andworkload analysis. In *Proceedings of the 3rd International Conference on Quantitative Evaluation of Systems (QEST'06)*. IEEE, Riverside, CA, USA, 119–120.

[8] Enrico Cambiaso, Gianluca Papaleo, and Maurizio Aiello. 2012. Taxonomy of slow DoS attacks to web applications. In *Proceedings of International Conference on Security in Computer Networks and Distributed Systems (SNDS)*. Springer, Trivandrum, India, 195–204.

[9] Cisco. 2017. Snort. https://www.snort.org/. (2017).

[10] Kristal Curtis, Peter Bodík, Michael Armbrust, Armando Fox, Mike Franklin, Michael Jordan, and David Patterson. 2010. *Determining SLO Violations at Compile Time.*

[11] Jeffrey Dean and Luiz André Barroso. 2013. The tail at scale. *Commun. ACM* 56, 2 (2013), 74–80.

[12] Mina Guirguis, Azer Bestavros, and Ibrahim Matta. 2004. Exploiting the transients of adaptation for RoQ attacks on Internet resources. In *Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP'04)*. IEEE, Berlin, Germany, 184–195.

[13] Robert "RSnake" Hansen. 2017. Slowloris HTTP DoS. https://web.archive.org/web/20090822001255/http://ha.ckers.org/slowloris/. (2017).

[14] Amir Herzberg and Haya Shulman. 2013. Socket overloading for fun and cache-poisoning. In *Proceedings of the 29th Annual Computer Security Applications Conference*. ACM, New Orleans, LA, USA, 189–198.

[15] Sabrina Hiller. 2015. Precise to the millisecond: NTP services in the "Internet of Things". https://www.retarus.com/blog/en/precise-to-the-millisecond-ntp-services-in-the-internet-of-things/. (2015).

[16] IETF. 2017. RFC 6298. https://tools.ietf.org/search/rfc6298/. (2017).

[17] Deepal Jayasinghe, Simon Malkowski, Qingyang Wang, Jack Li, Pengcheng Xiong, and Calton Pu. 2011. Variations in performance and scalability when migrating n-tier applications to different clouds. In *Proceedings of the IEEE International Conference on Cloud Computing (CLOUD'11)*. IEEE, Washington DC, USA, 73–80.

[18] Myeongjae Jeon, Yuxiong He, Hwanju Kim, Sameh Elnikety, Scott Rixner, and Alan L Cox. 2016. TPC: Target-Driven Parallelism Combining Prediction and Correction to Reduce Tail Latency in Interactive Services. In *Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, Atlanta, GA, USA, 129–141.

[19] Jaeyeon Jung, Balachander Krishnamurthy, and Michael Rabinovich. 2002. Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites. In *Proceedings of the 11th International Conference on World Wide Web*. ACM, Honolulu, Hawaii, USA, 293–304.

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

[21] Min Suk Kang, Soo Bum Lee, and Virgil D Gligor. 2013. The crossfire attack. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'13)*. IEEE, San Francisco, CA, USA, 127–141.

[22] Yu-Ming Ke, Chih-Wei Chen, Hsu-Chun Hsiao, Adrian Perrig, and Vyas Sekar. 2016. 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, Xi'an, China, 699–710.

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

[24] Ron Kohavi and Roger Longbotham. 2007. Online experiments: Lessons learned. *IEEE Computer Society* 40, 9 (2007).

[25] Aleksandar Kuzmanovic and Edward W Knightly. 2003. Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM'03)*. ACM, Karlsruhe, Germany, 75–86.

[26] Chien-An Lai, Josh Kimball, Tao Zhu, Qingyang Wang, and Calton Pu. 2017. milliScope: a Fine-Grained Monitoring Framework for Performance Debugging of n-Tier Web Services. In *Proceedings of the IEEE 37th International Conference on Distributed Computing Systems (ICDCS'17)*. IEEE, Atlanta, GA, USA, 92–102.

[27] Xiapu Luo and Rocky KC Chang. 2005. On a New Class of Pulsing Denial-of-Service Attacks and the Defense. In *Proceedings of Network and Distributed System Security Symposium (NDSS'05)*. San Diego, CA, USA.

[28] Gabriel Maciá-Fernández, Jesús E Díaz-Verdejo, Pedro García-Teodoro, and Francisco de Toro-Negro. 2007. LoRDAS: A low-rate DoS attack against application servers. In *Proceedings of International Workshop on Critical Information Infrastructures Security*. Springer, Málaga, Spain, 197–209.

[29] Microsoft. 2017. Microsoft Azure. https://azure.microsoft.com/en-us/?v=17.14. (2017).

[30] Jelena Mirkovic and Peter Reiher. 2004. A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review* 34, 2 (2004), 39–53.

[31] NSF. 2017. CloudLab. https://www.cloudlab.us. (2017).

[32] Brian J Odelson, Murali R Rajamani, and James B Rawlings. 2006. A new autocovariance least-squares method for estimating noise covariances. *Automatica* 42, 2 (2006), 303–308.

[33] Georgios Oikonomou and Jelena Mirkovic. 2009. Modeling human behavior for defense against flash-crowd attacks. In *Proceedings of the IEEE International Conference on Communications (ICC'09)*. IEEE, Dresden, Germany, 1–6.

[34] OW2. 2017. RUBBoS. http://jmob.ow2.org/rubbos.html. (2017).

[35] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. 2007. Survey of network-based defense mechanisms countering the DoS and DDoS problems. *ACM Computing Surveys (CSUR)* 39, 1 (2007), 3.

[36] PhantomJS. 2017. PhantomJS. http://phantomjs.org/. (2017).

[37] Pratap Ramamurthy, Vyas Sekar, Aditya Akella, Balachander Krishnamurthy, and Anees Shaikh. 2008. Remote Profiling of Resource Constraints of Web Servers Using Mini-Flash Crowds.. In *Proceedings of 2008 USENIX Annual Technical Conference*. Boston, MA, USA, 185–198.

[38] Supranamaya Ranjan, Ram Swaminathan, Mustafa Uysal, Antonio Nucci, and Edward Knightly. 2009. DDoS-shield: DDoS-resilient scheduling to counter application layer attacks. *IEEE/ACM Transactions on Networking (TON)* 17, 1 (2009), 26–39.

[39] Ryan Rasti, Mukul Murthy, Nicholas Weaver, and Vern Paxson. 2015. Temporal lensing and its application in pulsing denial-of-service attacks. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'15)*. IEEE, San Jose, CA, USA, 187–198.

[40] Huasong Shan, Qingyang Wang, and Qiben Yan. 2017. Very Short Intermittent DDoS Attacks in an Unsaturated System. In *Proceedings of the 13th International Conference on Security and Privacy in Communication Systems*. Springer, Niagara Falls, Canada.

[41] Qingyang Wang, Yasuhiko Kanemasa, Jack Li, Deepal Jayasinghe, Toshihiro Shimizu, Masazumi Matsubara, Motoyuki Kawaba, and Calton Pu. 2013. Detecting transient bottlenecks in n-tier applications through fine-grained analysis. In *Proceedings of the IEEE 33th International Conference on Distributed Computing Systems (ICDCS'13)*. IEEE, Philadelphia, PA, USA, 31–40.

[42] Qingyang Wang, Yasuhiko Kanemasa, Jack Li, Chien-An Lai, Chien-An Cho, Yuji Nomura, and Calton Pu. 2014. 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*. Broomfield, CO, USA.

[43] Qingyang Wang, Chien-An Lai, Yasuhiko Kanemasa, Shungeng Zhang, and Calton Pu. 2017. A Study of Long-Tail Latency in n-Tier Systems: RPC vs. Asynchronous Invocations. In *Proceedings of the IEEE 37th International Conference on Distributed Computing Systems (ICDCS'17)*. IEEE, Atlanta, GA, USA, 207–217.

[44] Yi Xie and Shun-Zheng Yu. 2009. Monitoring the application-layer DDoS attacks for popular websites. *IEEE/ACM Transactions on Networking (TON)* 17, 1 (2009), 15–25.

[45] Ying Xuan, Incheol Shin, My T Thai, and Taieb Znati. 2010. Detecting application denial-of-service attacks: A group-testing-based approach. *IEEE Transactions on parallel and distributed systems* 21, 8 (2010), 1203–1216.

[46] Chengxu Ye and Kesong Zheng. 2011. Detection of application layer distributed denial of service. In *Proceedings of the IEEE International Conference on Computer Science and Network Technology*, Vol. 1. IEEE, Harbin, China, 310–314.

[47] Jie Yu, Zhoujun Li, Huowang Chen, and Xiaoming Chen. 2007. A detection and offense mechanism to defend against application layer DDoS attacks. In *Proceedings of the IEEE 3rd International Conference on Networking and Services (ICNS'07)*. IEEE, Athens, Greece, 54–54.

[48] Saman Taghavi Zargar, James Joshi, and David Tipper. 2013. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE communications surveys & tutorials* 15, 4 (2013), 2046–2069.

[49] Ying Zhang, Zhuoqing Morley Mao, and Jia Wang. 2007. Low-Rate TCP-Targeted DoS Attack Disrupts Internet Routing.. In *Proceedings of Network and Distributed System Security Symposium (NDSS'07)*. San Diego, CA, USA.