Available online at www.sciencedirect.com
ScienceDirect
J. Parallel Distrib. Comput. 67 (2007) 318–335
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

Adversarial exploits of end-systems adaptation dynamics☆

Mina Guirguis[a,*], Azer Bestavros[b], Ibrahim Matta[b], Yuting Zhang[c]

[a]*Department of Computer Science, Texas State University, 601 University Drive, San Marcos, TX 78666, USA*
[b]*Department of Computer Science, Boston University, 111 Cummington Street, Boston, MA 02215, USA*
[c]*Department of Computer Science, Allegheny College, 520N. Main Street, Meadville, PA 16335, USA*

Received 7 August 2006; received in revised form 7 August 2006; accepted 25 October 2006

Abstract

Internet end-systems employ various adaptation mechanisms that enable them to respond adequately to legitimate requests in overload situations. Today, these mechanisms are incorporated in most scalable end-systems through the use of one or more component subsystems such as admission controllers, traffic shapers, content transcoders, QoS Controllers, and load balancers. While the design of these components has been heavily investigated and significantly fine-tuned for efficiency and scalability purposes, the security implication of the adaptation mechanisms used in these components has not been on the radar to system designers. To that end, this paper exposes adversarial exploits of the dynamics that result from the adaptive nature of these components. We show that a well orchestrated Reduction of Quality (RoQ) attack could induce significant inefficiencies or reduce the service quality of end-systems, without resorting to brute-force Denial-of-Service (DoS) exploits that target the limited steady-state capacity of these end-systems. We present a general analytical framework that captures the effect of RoQ exploits on the underlying optimization process of the adaptation mechanisms. Using detailed models, we instantiate this general framework for some of the aforementioned end-system adaptation mechanisms, focusing on admission controllers and load balancers. Our exposition is supported with numerical solutions of analytical models, which are validated using results from detailed simulations, and measurements from real Internet experiments performed in our lab.

Keywords: Security; Denial of Service; Scalable web services; Adaptive resource management; Performance evaluation

1. Introduction

End-system servers and networks (such as web servers and content delivery networks) have emerged as crucial building blocks of our current Internet infrastructure with profound impact on our economy and society. Due to the open nature of access to these end-systems, the designs of such systems have grown to be quite sophisticated to enable them to adapt adequately in response to the often erratic load offered by legitimate requests. However, as the complexity in these adaptation mechanisms increases, it becomes harder to understand their dynamic behavior.

1.1. The challenge of capturing/taming system dynamics

End-systems may exhibit elaborate dynamic behaviors due to resource management strategies in general (as in scheduling, load balancing, caching, etc.) and system adaptation strategies in particular (as in admission control, congestion control, etc.). These dynamics are quite hard to capture analytically or even empirically. As a result, models of computing system components often tend to abstract away such dynamics and focus instead on static properties obtained through aggregations over time scales that are long enough to hide the transients of adaptation; metrics used to monitor and evaluate a system's performance (such as utilization, delay, jitter, and admission rates) are typically expressed as *shapeless* mean values, which do not give us insights into the inefficiencies caused by transients over time scales shorter than those used in measuring such metrics. Such relatively little attention by computing system designers and practitioners to system dynamics stands in sharp contrast to how other engineered systems, such as the electric grid or

☆ This work was supported in part by a number of NSF awards, including CNS Cybertrust Award #0524477, CNS NeTS Award #0520166, CNS ITR Award #0205294, and EIA RI Award #0202067, and by grants from Fortress Technologies.
* Corresponding author.
E-mail address: msg@txstate.edu (M. Guirguis).

0743-7315/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2006.10.005

mechanical artifacts, are evaluated. For such systems, the characterization of system dynamics is front and center to protect against oscillatory behaviors and instabilities.

System dynamics could be "safely ignored" (or abstracted out as we like to say in Computer Science), if one can ensure that such dynamics will not interfere, or that they will have negligible impact on the overall performance of the system, which is typically measured using metrics that gauge efficiency and responsiveness. Such assurances are warranted for closed systems with predictable, non-adversarial workloads. However, for open systems, such dynamics cannot be "safely ignored" as they could be exploited by adversaries. Indeed, the main goal of this paper is to show that such exploits are not only plausible, but that their impact could be significant. Notice that while system dynamics could be shown not to interfere with, or significantly impact the fidelity of an end-system under non-adversarial (even if bursty or erratic) workloads, the same could not be said for adversarially engineered workloads.

1.2. Adversarial exploits of end-system adaptation mechanisms

To deal with the open access nature of the Internet and the potential overload situations that may arise, end-systems typically employ a combination of (1) admission controllers, (2) content adaptation controllers and (3) load balancers in order to maintain high-fidelity operation. Admission controllers ensure that the overall offered load to the system, over a long-enough timescale, does not exceed its rated capacity. Content adaptation controllers mitigate the overload conditions by reducing the quality of the serviced content, using content transcoding for example. Load balancers, on the other hand, ensure that the system capacity and performance is achieved through a judicious distribution of load to the servers and resources available in the system.

In this paper we show that a determined adversary could bleed a system's capacity or significantly reduce its service quality by subjecting it to a fairly low-intensity (but well orchestrated and timed) request stream that causes the system to become very inefficient, or unstable. While in [20] we gave an example of such attacks—which we termed Reduction of Quality (RoQ; as in "rock") attacks—on Internet resources employing Active Queue Management (AQM) schemes, in this paper, we consider RoQ exploits on adaptation mechanisms employed in end-systems. In particular, this paper combines our work on RoQ exploits on admission controllers [21] and on load balancers [22], and briefly discusses RoQ exploits on content adaptation controllers.

1.3. An illustrative RoQ exploit

Current adversarial strategies for Denial of Service (DoS) are *brute force* [9]. An attacker may render a system, say a Web server, useless by subjecting that system to a sustained attack workload (e.g., syn attack) that far exceeds that system's capacity. The result is that legitimate requests experience a much degraded response from a persistently overloaded system—or even are denied access to that system altogether. Could an attacker achieve similar outcomes without overloading the system in a persistent manner? The answer is yes. To explain why this is the case, we give a simple illustrative example.

Consider an admission controller that sets its admission rate of incoming requests as a function of the utilization of its back-end system [46,47,6]. Now, consider a point in time when the offered load is low enough for the admission controller to allow a large percentage of all requests to go through. At this point, a surge in demand (e.g., a large number of requests) in a very short period of time would push the system into overload. This, in turn, would result in the admission controller shutting off subsequent legitimate requests for a long time—given the fact that under overloaded conditions, the system operates in an inefficient region (e.g., due to thrashing). Once the system "recovers" from the ill-effects of this unsuspected surge in demand, an attacker would simply repeat the process. Albeit simplistic, this attack illustrates how adaptation strategies may be exploited by adversaries to reduce system's fidelity.

Dynamic content adaptation and load balancing controllers could be targets of RoQ exploits as well. In these settings, an attacker's goal would be to reduce the quality of the content or to increase the response time for legitimate requests. Since users may not be willing to wait for results, or tolerate a degraded content (e.g., a degraded quality video stream), RoQ exploits in such settings could translate into a denial of service, similar to the admission control example above.

1.4. Motivation behind mounting RoQ exploits

We believe the work presented in this paper captures an ongoing trend of launching attacks that are more stealthy, with the main goal of the attackers not to be discovered. For example, the work presented in [26], shows how attackers are moving away from bandwidth floods to attacks that mimic the web browsing of a larger set of clients. There are two main reasons of why an attacker would prefer to launch a RoQ attack over a traditional DoS attack. First, it takes a lot of effort to mount a DoS attack; an attacker needs to find the machines (zombies) that are going to be used and needs to control them to direct their malicious attack traffic towards the victim. Both of these tasks are hard to accomplish. Second, DoS attacks, once launched, are very easily discovered. Thus, proper defense mechanisms could be set in place very quickly, to mitigate the attack's impact and to trace-back the preparators. RoQ attacks, on the other hand, do not require a lot of resources to mount the attack (the cost in mounting the attack is taken explicitly into account) and it is very hard to tell if a system is under attack in the first place (the dynamics resulting from a RoQ attack, could still occur under normal operation), hence, detection and trace-back are that much harder than under DoS attacks.

1.5. Paper outline

Section 2 summarizes the premise and the definition of RoQ attacks with a focus on the notion of attack potency to

quantify the impact of RoQ attacks. We also present an analytical model, whereby the transients of adaptation are simply the result of an optimization process which forces an end-system to converge to a stable operating point. Under such model, one could view a RoQ attack as a persistent attempt to regularly knock the system off its stable (or quiescent) operating point. In Sections 3 and 4, we present dynamic models along with results obtained from simulation and Internet experiments, for an admission controller and a load balancer, respectively. In Section 5, we outline few challenges for the detection and mitigation of RoQ exploits. We outline possible defense mechanisms. In Section 6, we briefly discuss related work, noting that throughout this paper, we point to various pieces of research work as appropriate. We conclude in Section 7 with a summary and with future directions.

2. RoQ attack premise and definition

This paper leverages some of the models and analysis of RoQ attacks presented in earlier work of ours [20]. In this section, we briefly review the premise of RoQ attacks, emphasizing its novel conception of the attacker's goal: namely to maximize damage per unit attack load (or cost). We then illustrate the general framework whereby the transients of adaptation are the result of an optimization process which forces the system's operation into a stable regime.

2.1. Attack goal and definition

We consider a RoQ attack comprising a burst of M requests sent to a system element at the rate of δ requests per second over a short period of time τ, where $M = \delta\tau$. This process is repeated every T units of time. We call M the *magnitude* of the attack, δ the *amplitude* of the attack, τ the *duration* of the attack, and T the *period* of the attack.

For the above RoQ attack, we define Π, the *attack potency*, to be the ratio between the *damage* caused by that attack and the *cost* of mounting such an attack. Clearly, an attacker would be interested in maximizing the damage per unit cost—i.e., maximizing the attack potency.

$$\text{Potency} = \Pi = \frac{\text{Damage}}{\text{Cost}^{\frac{1}{\Omega}}}. \tag{1}$$

The Potency definition given by Eq. (1) does not specify what constitutes "damage" and "cost". Clearly, one may consider various instantiations of these metrics. For example, for an attacker aiming to minimize a web server availability, through exploiting its admission controller, a natural metric of "damage" would be the difference between the total number of requests admitted before and after the attack (excluding the attacker's requests). If the attacker aims to maximize the jitter in the users' observed response time, then a natural metric of "damage" would be the difference between the standard deviation of the time it takes to process a request before and after the attack. Similarly, there could be a number of different metrics for what constitutes "cost". Examples include the effective attack request-rate (i.e., M/T), the attack amplitude δ, the attack duration τ, etc.

The Potency definition given by Eq. (1) uses a parameter Ω to model the aggressiveness of the attacker. A large Ω reflects the highest level of aggression, i.e., an attacker bent on inflicting the most damage and for whom cost is not a concern. Mounting a DoS attack is an example of such behavior. A small Ω reflects an attacker whose goal is to maximize damage with minimal exposure. Thus, the parameter Ω enables us to identify families of attacks based on aggression. Throughout this paper, we take Ω to be 1, where we compare "damage" versus "cost" directly.

2.2. Adaptation as an optimization process

We consider a system subjected to multiple request streams, each of which offers a load characterized by a rate x_r of requests. In a web server setting, x_r would represent the request rate for a particular service r (e.g., in hits/s). The value of x_r is adapted based on feedback received from the system (equivalently, prices). In a web server setting, that pricing feedback would be the request response time, which is a function of resources consumed (e.g. CPU, disk and memory.) [1]

The adaptation of x_r would typically follow differential equations (2) where $\mathcal{I}(.)$ and $\mathcal{D}(.)$ represent the increase and decrease functions, which depend on the rates $x(t)$ and the function $p_l(.)$ reflecting the prices/costs fed back to the source as the input load on the resources l used by stream r varies.

$$\frac{d}{dt}x_r(t) = \mathcal{I}(x(t), p_l(x(t))) - \mathcal{D}(x(t), p_l(x(t))). \tag{2}$$

In analyzing the convergence of such system to steady-state rates x_r^*, we resort to optimal control theory to show that the evolution of the system leads to optimizing some objective function, called the system's Lyapunov function [38]. The basic idea is to find such Lyapunov function $U(x)$ of the system state x that is positive, continuous and strictly concave, such that $\frac{d}{dt}U(x(t)) > 0$ if $x_r(t) \neq x_r^*$ and equals zero when $x_r(t) = x_r^*$ for all r.

Lyapunov function $U(x)$ is generally of the form in Eq. (3), where the first term represents the gain in request rates and the second term represents the associated costs (prices). Thus by optimizing $U(x)$ the system optimizes its net gain.

$$U(x) = \sum_r \mathcal{G}(x_r) - \sum_l \mathcal{C}(p_l(x)). \tag{3}$$

Given that the system converges to a fixed point x_r^*, one would be interested in the rate of convergence as this will determine the speed with which transients subside. An optimized RoQ exploit would leverage such transients of adaptation to knock the system off whenever it is about to stabilize. Let μ determines the rate of convergence of the system—a higher value indicates faster convergence. Notice that for a linearized system, in the form

[1] While we give an example of what constitutes a price in a specific setting, other pricing functions are certainly possible.

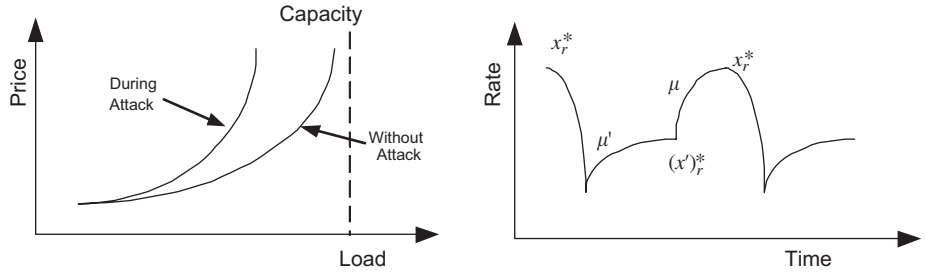


Fig. 1. An example of a web server pricing function as the load on the system varies. RoQ attacks, effectively, will keep the pricing function changing (left) and Effect of a RoQ attack pattern on the request rate ($x_r(t)$). RoQ attacks will hinder convergence to steady state points (right).

of $\dot{y} = Ly$ where L is a matrix and y is a vector of state variables, the smallest eigen value of L determines the rate of convergence, μ.

The analysis we have conducted so far could be used to provide insights into the effect of adversarial attacks that aim to exploit the optimization process that leads the system to converge to steady-state rates x_r^*. We do so next.

Assume that the system had already stabilized to its steady-state x_r^* values. Since a resource is used to its almost maximum capacity, the additional attack load is likely to push the resource towards saturation where the fed-back prices are extremely high—see Fig. 1(left). Since the RoQ attack involves a sustained rate of δ for τ units of time, the system will be pushed to a new stable point, say $(x')_r^*$. Let μ' refer to the new rate of convergence to the new stable point (i.e., from x_r^* to $(x')_r^*$). Since the capacity of the attacked resource is effectively reduced during the attack duration τ, the resource pricing function is pushed to the left, as shown in Fig. 1(left). Such higher prices result in faster convergence (i.e., higher μ') and lower $(x')_r^*$.

As soon as the system stabilizes to $(x')_r^*$, an optimized RoQ exploit would cease, allowing the system to return to its original state x_r^*. This pattern then repeats as illustrated in Fig. 1(right), in effect forcing the system to spend its time oscillating between different states, due to the presence and absence of the attack traffic. Note that in general, the attack traffic may destroy the "contractive" mapping property of the pricing function and so the system may not converge to a fixed point while under attack.

Having defined the RoQ exploit, we now turn our attention to assessing its potency as defined in Eq. (1). With respect to our analytical model parameters, one may capture the "damage" caused by the attack using the expression $\delta(\frac{1}{\mu'} + \frac{1}{\mu})$. Intuitively, this expression represents the wasted capacity (or other service qualities such as delay and rate jitter, as we discuss later) during instability. Also, one may capture the "cost" of the attack by $(\delta/(\frac{1}{\mu'} + \frac{1}{\mu}))$. Intuitively, the cost increases with increasing the attacker's peak rate and decreases with longer attack period. Again, we emphasize that the definition of potency allows for many other instantiations of "damage" and "cost" (which may be more meaningful as we will do later in the paper) and that our specific choices above are for illustrative purposes.

Accordingly, we calculate potency using Eq. (4), where Ω reflects the relative values that an attacker attributes to "damage" versus "cost", or equivalently the desired level of aggression

$$\Pi = \frac{\delta\left(\frac{1}{\mu'} + \frac{1}{\mu}\right)}{\left(\delta/\left(\frac{1}{\mu'} + \frac{1}{\mu}\right)\right)^{1/\Omega}} = \delta^{1-\frac{1}{\Omega}}\left(\frac{1}{\mu'} + \frac{1}{\mu}\right)^{1+\frac{1}{\Omega}}. \quad (4)$$

In the next two sections, we will consider more elaborate analytical models to gain further insights into more complicated adaptation dynamics of specific systems. In Section 3, we consider RoQ exploits on admission controller, and in Section 4 we consider RoQ exploits on dynamic load balancers.

3. RoQ attacks on admission controllers

3.1. Adaptation through admission control

Admission controllers—a common fixture of computing systems and networks—are used to protect against overload conditions by rejecting requests (or offered load) that would push a system beyond a quiescent operating point. Admission control strategies are employed in operating systems, database servers, real-time and multimedia servers, among many others. As the example in Section 1 illustrated, admission controllers may be targets of RoQ exploits. In this section, we instantiate from the general model we presented in Section 2, a detailed model for studying the vulnerabilities of admission controllers.

3.2. Model derivation

The operation of a server (say a web server) whose load is regulated by an admission controller is modeled by three components: the *admission controller*, the server, and the *feedback monitor*.

The admission controller determines the percentage of requests that should be accepted (i.e., admitted) for service. This *admission rate* is based on the deviation of the server's state (utilization) from a desirable *set value*. We use a PI controller [38] to translate the *error signal* (deviation in utilization from a set value) to an admission rate. The impact of using other forms of controllers can also be studied using this same framework. For instance, one can think of an Additive-Increase Multiplicative-Decrease (AIMD) admission controller. AIMD admission control would shut off admission rate (when system gets to overload) exponentially but would only increase it,

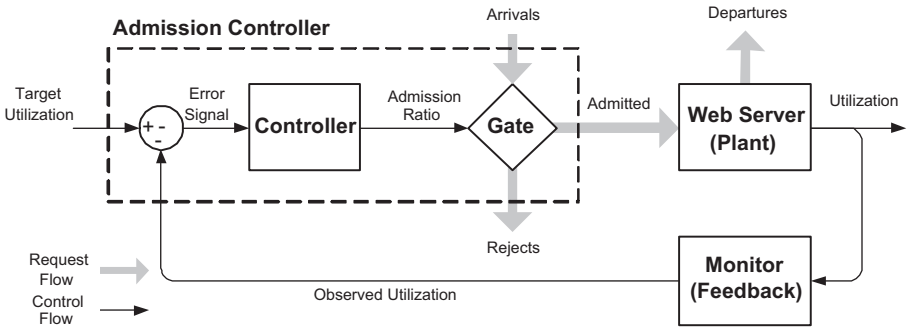


Fig. 2. Block diagram showing the various components of the admission control feedback loop for a web server (as an example of an Internet end-system).

Table 1
Parameters of the linearized model used to analyze potency of RoQ exploits of PI admission control

Parameter	Description
$\alpha(.)$	Admission ratio
$\rho(.)$	Server utilization
$n(.)$	Number of requests pending inside the system
$\lambda(.)$	Rate of arrival of requests
$m(.)$	Number of requests admitted
K	PI controller constant
ρ^*	Target server utilization
A, B, C, D, N	Constants describing load/utilization curve
ρ_o	Server utilization beyond which the server thrashes
ω	Thrashing index
μ_{max}	Maximum service rate
μ_{min}	Minimum service rate
δ	Attack rate
τ	Attack duration
T	Attack period

when the system is under-loaded, linearly. Admitted requests are then processed by the server. The feedback monitor periodically measures the server's utilization and reports back a value thereof (e.g., average over a time interval or EWMA) to the admission controller. This feedback control system is depicted in Fig. 2.

Table 1 summarizes the notation and the description of the parameters used in our model.

Instantiating our general model of Section 2, the pricing function of the *admission controller* is given by the relationship between the admission ratio (equivalently, prices) of web requests $\alpha(.)$ and the utilization of the server $\rho(.)$ (load). The latter is a function of the current total number of requests pending inside the system $n(.)$, which in turn evolves as a function of both the admission rates of requests $m(.)$ and the service rate of the web server. Fig. 3 shows two specific (simple) examples of the relationships between $n(.)$ and $\rho(.)$, and between $\rho(.)$ and the web server (plant) service rate.

A natural goal of a RoQ exploit on an admission controller is to maximize the damage caused by a periodic adversarial attack of magnitude M, where damage constitutes the reduction in the number of legitimate requests admitted to the system per attack period, or equivalently the difference between the admission rate under normal conditions and that achieved when the exploit is mounted. Let R_w denote the number of rejected requests due to a single periodic attack with parameters $M = \delta \times \tau$, over an attack period, T. Thus, the attack potency $\Pi = R_w/M$.

As in the generic analytical model of Section 2, the attack traffic can effectively reduce the service rate of the resource by pushing the system into high utilization, leaving the system in a "thrashing" mode of operation where it takes a long time to recover.

Considering a discrete-time model, Eq. (5) represents a PI controller, where the admission ratio, $\alpha(i)$, at time i, is updated based on the error signal between the target utilization, ρ^*, and the current utilization, $\rho(i)$. K is the PI controller's constant, which plays an important role in how aggressive the controller reacts to the error signal. In particular, a higher value of K will tend to cause the admission controller to react more aggressively to the error signal, but possibly causing transients in the utilization to be reflected in the admission ratio. A lower value of K will tend to achieve higher stability margins, but possibly causing the admission controller to be less responsive to sudden changes in utilization.

$$\alpha(i) = K \times (\rho^* - \rho(i)) + \alpha(i-1). \tag{5}$$

Eq. (6) represents the utilization, $\rho(.)$ as a function of the number of requests pending inside the system. N, A, B, C and D are constants [2]

$$\rho(i) = \begin{cases} An(i) + B & n(i) < N, \\ \min[Cn(i) + D, 1] & \text{otherwise.} \end{cases} \tag{6}$$

Notice that $\rho(.)$ has a lower bound of B, which represents the utilization when the offered load is zero, reflecting the utilization of the system due to background operating-system services, etc. When $n(i) < N$, the server operates efficiently; its utilization increases slowly in proportion to $n(i)$ as dictated by the (small) constant A. Beyond N, utilization increases quickly in proportion to $n(i)$ as dictated by the constant $C > A$, until it reaches its upper bound of 1.

[2] Since $\rho(.)$ is continuous, three constants suffice to describe Eq. (6), but to simplify the notation, we use four constants (A, B, C and D).

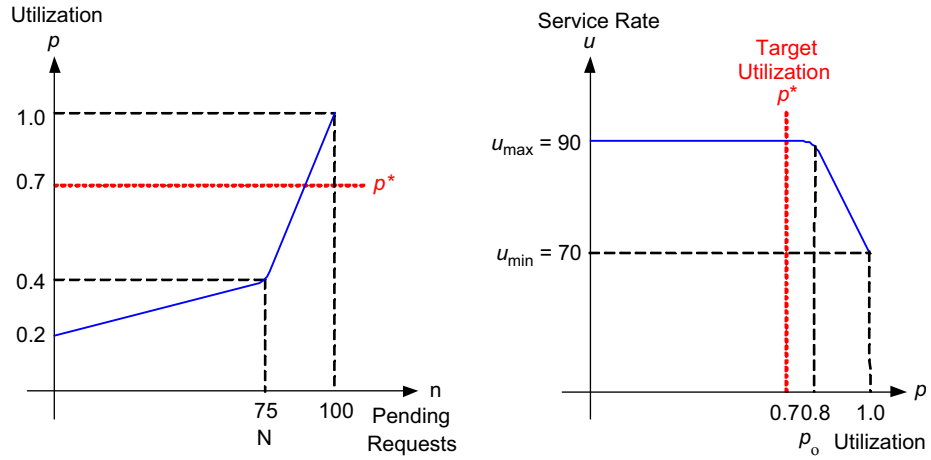


Fig. 3. Instances of (linearized) pricing functions showing the relationships often observed between load and utilization (left) and utilization and thrashing expressed as degradation in service rate (right).

Given the admission rate $\alpha(i)$ and arrivals $\lambda(i)$ at time i, the number of requests admitted at time i is given by $m(i) = \alpha(i) \times \lambda(i)$. Notice that this adaptation of $m(.)$ represents a Multiplicative-Increase Multiplication-Decrease (MIMD) policy since $\lambda(.)$ is simply multiplied by the price $\alpha(.)$. This leads us to Eq. (7) for the evolution of the number of pending requests $n(.)$

$$n(i) = n(i-1) + m(i) - (\mu_{\max} - I(\rho(i) > \rho_o) \times \omega(\rho(i) - \rho_o)), \tag{7}$$

where $I(x)$ is the indicator function that equals 1 if condition x is true, and ω is the *thrashing index*, a constant that represents the severity of degradation in service rate as $\rho(.)$ increases beyond ρ_o. Eq. (7) implies that as long as $\rho(.)$ is less than ρ_o, the system is operating at its rated capacity. However, once it gets into overload, its capacity is reduced.

We assess the vulnerabilities of the above-modeled admission controller to a RoQ exploit comprising periodic bursts of δ requests/s sent over a short duration τ, with a period T. For simplicity, we assume that $\tau = 1$.

Let $\alpha(0)$ denote the admission rate for a steady-state (constant) arrival rate of λ prior to an attack starting at time 0. Assuming that δ is of a high-enough value to force $\rho(.)$ to reach unity, we get a constant error signal of $\rho^* - 1$. Thus the PI controller (cf. Eq. (5)) will start decreasing the admission rate in fixed steps of $K(\rho^* - 1)$. One can easily see that at time i, $\alpha(i) = K(\rho^* - 1)i + \alpha(0)$. Clearly, $\rho(.)$ will remain at 1 until the load is low enough to decrease ρ to a value lower than unity. Let us denote by θ the period of time during which $\rho(.)$ remains at one.

During θ, the total number of requests admitted to the system will include whatever was admitted from the attack traffic, plus whatever was admitted from regular arrivals, based on the changing values of the PI controller—namely, $\delta\alpha(0) + \lambda(\alpha(0) + \alpha(1) + \cdots + \alpha(\theta))$. During that time, since ρ is stuck at one, the departure rate is equal to $\mu_{\max} - \Omega(1 - \rho_o)$ which we denote by $\mu_{\min}$—the minimum departure rate. Thus θ can be expressed as

$$\theta = \frac{\delta\alpha(0) + \lambda(\sum_{i=0}^{\theta} \alpha(i))}{\mu_{\min}}. \tag{8}$$

Notice that the summation $\sum_{i=0}^{\theta} \alpha(i)$ is equal to

$$\sum_{i=0}^{\theta} \alpha(i) = (\theta + 1)\alpha(0) + K(\rho^* - 1)\frac{\theta}{2}(\theta + 1). \tag{9}$$

This allows us to derive a second-order equation which can be solved for (the positive value of) θ and is given by

$$\left(\frac{\lambda K(1-\rho^*)}{2}\right)\theta^2 - \left(\lambda\alpha(0) - \frac{K(1-\rho^*)}{2} - \mu_{\min}\right) \times \theta - (\lambda + \delta)\alpha(0) = 0. \tag{10}$$

Since $\frac{K(1-\rho^*)}{2}$ is typically relatively small compared to $\mu_{\min}$ and $\lambda\alpha(0)$, one could approximate the above equation by [3]

$$\left(\frac{\lambda K(1-\rho^*)}{2}\right)\theta^2 - (\lambda\alpha(0) - \mu_{\min})\theta - (\lambda+\delta)\alpha(0) = 0. \tag{11}$$

Notice that the attacker's traffic δ only appears in the constant coefficient of the above second-order equation. This means that as the attacker's traffic increases, the positive root of the above equation will be larger, resulting in a larger value of θ, implying a longer time for the server to fully react to the attack. This can be easily seen when we solve for the roots of the second-order equation of the form $a\theta^2 + b\theta + c = 0$; the term $b^2 - 4ac$ is always positive since c is negative and it gets larger as the magnitude of c gets larger, so the value of the positive root increases.

During θ, and as the system reacts to the overload caused by the burst of attack traffic, the admission controller would have

[3] The numerical solution in the absence of this approximation matches very closely the closed-form solution given in Eq. (11).

rejected R_θ (legitimate) requests. The value of R_θ can be easily derived using Eq. (9):

$$\begin{aligned} R_\theta &= \lambda\{(\alpha(0)-\alpha(1))+(\alpha(0)-\alpha(2))+\cdots+(\alpha(0)-\alpha(\theta))\} \\ &= \lambda\left(\theta\alpha(0)-\sum_{i=1}^{\theta}\alpha(i)\right) \\ &= \lambda\left(K(1-\rho^*)\frac{\theta(\theta+1)}{2}\right). \end{aligned} \tag{12}$$

Beyond θ, ρ decreases precipitously as a result of the admission controller's reaction to the overload caused by the RoQ exploit. Eventually, this will cause the PI controller to reverse course by increasing the admission rate so that ρ can reach ρ^*. This increase in ρ will span two epochs of time ϕ_1 and ϕ_2, corresponding to whether the number of pending requests is less than N or larger than N, respectively. Thus, after time $\theta+\phi_1+\phi_2$, the admission rate will once again reach $\alpha(0)$, which represents the initial condition before the attack was waged.

Next, we derive how long it takes the system to again get its admission rate *close enough* to $\alpha(0)$. This period is divided into two regions as dictated by the piecewise linear function of load on utilization. We start by calculating the admission rate starting from $\alpha(\theta)$. At time $\theta+1$, the admission rate $\alpha(\theta+1)$ is given by

$$\begin{aligned} \alpha(\theta+1) &= Ke(\theta)+\alpha(\theta) \\ &= K(\rho^*-\rho(\theta))+\alpha(\theta) \\ &= K(\rho^*-(An(\theta)+B))+\alpha(\theta) \\ &= K(\rho^*-(A\alpha(\theta)\lambda+B))+\alpha(\theta) \\ &= K(\rho^*-B)+\alpha(\theta)(1-KA\lambda) \\ &= K_1+K_2\alpha(\theta), \end{aligned} \tag{13}$$

where $e(\theta)=\rho^*-\rho(\theta)$ is the error signal to the PI controller and K_1 and K_2 are given by $K(\rho^*-B)$ and $(1-KA\lambda)$, respectively. In the above derivation steps of Eq. (13), since the attack had subsided and the admission ratio is at its lowest value, whatever requests get admitted in one time step are served by the end of this time step; thus the number of pending requests $n(.)$ is simply given by the number of admitted requests $m(.)=\alpha(.)\lambda$.

Thus, at any time i after θ, the admission rate is given by

$$\begin{aligned} \alpha(\theta+i) &= K_1\frac{1-K_2^i}{1-K_2}+K_2^i\alpha(\theta) \\ &= \frac{K_1}{1-K_2}+K_2^i\left(\alpha(\theta)-\frac{K_1}{1-K_2}\right). \end{aligned} \tag{14}$$

Let ϕ_1 denote the time it takes for the admission rate to recover to some value $\hat{\alpha}(0)$, which is the point in time when utilization switches functions based on N. Solving for ϕ_1

$$\phi_1=\log_{K_2}\left(\frac{\hat{\alpha}(0)(1-K_2)-K_1}{\alpha(\theta)(1-K_2)-K_1}\right). \tag{15}$$

Once the number of pending requests exceeds N, at time ϕ_1, the admission rate $\alpha(\theta+\phi_1+1)$ is given by

$$\begin{aligned} \alpha(\theta+\phi_1+1) &= Ke(\theta+\phi_1)+\alpha(\theta+\phi_1) \\ &= K_3+K_4\alpha(\theta+\phi_1). \end{aligned} \tag{16}$$

Similarly, it can be easily shown that K_3 and K_4 are given by $K(\rho^*-D)$ and $(1-KC\lambda)$, respectively. Let ϕ_2 denote the time it takes for the admission rate to recover to some value $\tilde{\alpha}(0)$, which is close to $\alpha(0)$, starting from $\hat{\alpha}(0)$. Solving for ϕ_2

$$\phi_2=\log_{K_4}\left(\frac{\tilde{\alpha}(0)(1-K_4)-K_3}{\hat{\alpha}(0)(1-K_4)-K_3}\right). \tag{17}$$

We are now ready to compute the number of (legitimate) requests which are rejected as a result of the RoQ attack (in a single period T). The total number of rejected requests are those rejected during $\theta+\phi_1+\phi_2$. During the period ϕ_1, the total number of rejected requests R_{ϕ_1} is given by

$$\begin{aligned} R_{\phi_1} &= \lambda\{(\alpha(0)-\alpha(\theta+1))+\cdots+(\alpha(0)-\alpha(\theta+\phi_1))\} \\ &= \lambda\left\{\phi_1\alpha(0)-\sum_{i=1}^{\phi_1}\alpha(\theta+i)\right\} \\ &= \lambda\left\{\phi_1\alpha(0)-\sum_{i=1}^{\phi_1}\left\{\frac{K_1}{1-K_2}+K_2^i\left\{\alpha(\theta)-\frac{K_1}{1-K_2}\right\}\right\}\right\} \\ &= \lambda\left\{\phi_1\alpha(0)-\frac{\phi_1K_1}{1-K_2}-\sum_{i=1}^{\phi_1}\left\{K_2^i\left\{\alpha(\theta)-\frac{K_1}{1-K_2}\right\}\right\}\right\} \\ &= \lambda\left\{\phi_1\alpha(0)-\frac{\phi_1K_1}{1-K_2}-\left\{\alpha(\theta)-\frac{K_1}{1-K_2}\right\}\sum_{i=1}^{\phi_1}K_2^i\right\} \\ &= \lambda\left\{\phi_1\alpha(0)-\frac{\phi_1K_1}{1-K_2}-\left\{(\alpha(0)-K(1-\rho^*)\theta)\right.\right. \\ &\quad \left.\left.-\frac{K_1}{1-K_2}\right\}\left\{K_2\frac{1-K_2^{\phi_1}}{1-K_2}\right\}\right\}. \end{aligned} \tag{18}$$

Similarly, during ϕ_2, the total number of rejected requests R_{ϕ_2} is given by

$$\begin{aligned} R_{\phi_2} &= \lambda\left\{\phi_2\alpha(0)-\frac{\phi_2K_3}{1-K_4}\right. \\ &\quad \left.-\left\{\frac{N}{\lambda}-\frac{K_3}{1-K_4}\right\}\left\{K_4\frac{1-K_4^{\phi_2}}{1-K_4}\right\}\right\}. \end{aligned} \tag{19}$$

The above results can be used to calculate the potency of the attack, which is given by

$$\pi=\frac{R_\theta+R_{\phi_1}+R_{\phi_2}}{M}. \tag{20}$$

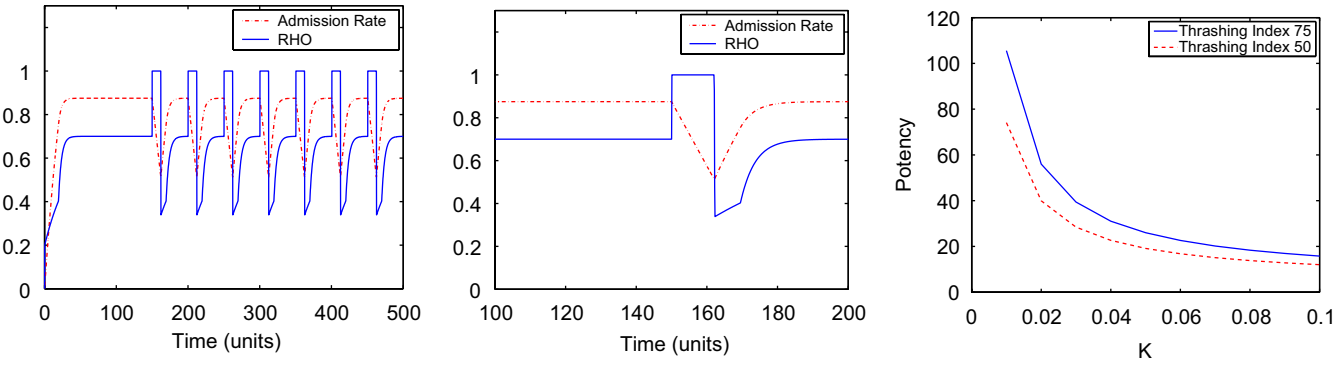


Fig. 4. Assessment of vulnerability to RoQ attacks: results from the numerical model (left and center) and the effect of K on the Potency (right).

3.3. Numerical solution

We numerically solve the above system. We assume that legitimate requests arrive at a rate λ of 100 requests per unit time (1 s). The number of pending requests $n(.)$ drives $\rho(.)$ as shown in Fig. 3(left), with constants $A = 0.00267$, $B = 0.2$, $C = 0.024$, $D = -1.4$, $K = 0.01$ and $N = 75$. The service rate $\mu(.)$ is driven by $\rho(.)$ as shown in Fig. 3(right), with $\mu_{\max} = 90$, $\mu_{\min} = 70$, $\rho_o = 0.8$, and $\omega = 70$. Fig. 4(left) shows the results we obtained. It shows the admission rate as well as the utilization of the back-end system over time. Clearly, within 50 s of operation, the system converges to an efficient operating region with admission rate at 0.875 and utilization is around its target value of 0.7. The RoQ attack starts at time 150 for a duration of $\tau = 1$ s and is repeated every $T = 50$ s, producing a potency of over 100, i.e., one request from the attacker results in over 100 legitimate requests being denied service. Fig. 4(centre) takes a closer look at the period of time between 100 and 200. It is clear that the admission ratio drops to below 0.6 from its steady-state value of 0.875.

Clearly, potency depends largely on the choice of the admission controller parameter, K, as well as the degradation in the service rate, $\mu_{\min}$. Fig. 4(right) show this dependence by showing the attack potency as a function of K for different values of ω. These plots were obtained using Eq. (20) and the closed-form solutions for R_θ (from Eq. (12)), R_{ϕ_1} (from Eq. (18)) and R_{ϕ_2} (from Eq. (19)). Clearly, the value of K is critical as it reflects the sensitivity of the PI controller to the error signal. It exposes the tradeoffs between efficiency and tolerance to RoQ exploits. A larger value of K will lower the potency of a given attack. However, a large value of K implies that the system will react swiftly to minor changes in its workload. Under normal operations, this is quite undesirable as it compromises stability. In Section 5, we address this issue along with the possibility of adjusting such parameters on-line as a possible defense mechanism.

The model developed above assumes a feedback delay of unit value—i.e., the utilization at time i drives the admission decision at time $i + 1$. In a practical setting, this feedback delay will slow the controller's reaction making it even more vulnerable. We illustrate the impact of feedback delay in our experiments.

3.4. Internet experiments

We assess the impact of RoQ attacks on an admission controller that lies in front of a Linux web server through real experiments performed in our lab. We conduct a series of experiments to investigate the effect of different values of the control parameter K in the presence and absence of feedback delay. Due to the wide deployment of web servers in the current Internet, we used a web server as an example of an end-system functionality that is subject to admission control. We could have instead used an application server or a database server. [4] In our experiments and similar to our analysis, we used a simple PI controller to derive the admission ratio. Other controllers, such as AIMD controller, could have been used as potential targets for RoQ attacks.

Notice that our main goal is to give evidence that such attacks could be carried out, as opposed to fully characterizing the possible damage that could be inflicted. A full characterization of damage is hard to assess due to the following two limitations:

(1) When faced with severe thrashing, Linux, on which the web server in our experiments runs, starts killing threads to get itself out of thrashing. This behavior, while acceptable for simple HTTP transactions (such as file transfer) is not acceptable in other more realistic and prevalent scenarios—e.g., when HTTP sessions are involved, and when backend system state could be compromised. This is precisely why the analytical model we used in the previous section assumed that the admission controller is the only entity that is responsible for admitting and rejecting requests and that it does not "kill" threads if the system is thrashing. This is important to ensure thread integrity in

[4] Indeed, such systems would be much more vulnerable to RoQ attacks by virtue of the more granular nature of their services, and hence their ability to recover from overload conditions.

many scenarios. For example, a database server with too many admitted transactions, cannot simply kill web server threads to get rid of thrashing. This may result in a violation of the system's integrity which, in return, could impose longer periods of recovery. [5]
Linux' interference with our admission controller (by imposing its own admission control) makes it impossible for us to assess the true impact of a RoQ attack once Linux starts its own thrashing prevention measures. Thus our experiments were only carried out under moderate load levels—i.e., at thrashing levels that did not trigger Linux's thread-killing measures. This naturally puts an upper-limit on the achievable potency values that we are able to observe and reliably measure in our laboratory experiments. In other words, the results we report should be viewed as lower bounds on the achievable potencies. Indeed, they will tend to be lower than those predicted analytically, since our models did not account for the existence of an outside mechanism (namely Linux' behavior in overloads) that "clears" thrashing.

(2) Httperf [36], which we use to generate HTTP requests, is not a perfect open-loop load generator. In particular, a client cannot open new connections when there are more than 1024 requests pending (this is an operating-system limitation on the number of open file descriptors). Having more machines to generate requests could indeed simulate a "more open-loop" system. In our experiments, we used four machines to generate requests.

Given the above two limitations, the potency values reported next are lower than those obtained analytically, and lower than what would be possible in a real setting—they should be interpreted as lower bounds; potencies of real attacks could well be much higher.

3.4.1. Experimental setup

Fig. 5 depicts the experimental setup we used in running our experiments. It consists of a server machine (S) running Minihttpd [34] web service and four client machines (C_1, C_2, C_3 and C_4) generating web traffic. We have implemented an admission controller to the web server, where requests from the clients are admitted to the server based on a dynamically adjusted admission ratio. The clients and the server are connected through a 2-hop switched LAN with 100 Mbps link capacities. All machines run Linux 2.4.20. The clients use Httperf [36] to generate their web traffic.

A client (C_i) is configured to send cgi requests to the server (S) through HTTP 1.0. Upon an arrival of a request, the server admits or rejects the request based on the admission ratio. For each admitted request, the server forks a new thread, which executes a cgi script. Each cgi script accesses 1 Mbit of memory and returns 4 Kbit of data back to the client. Under normal conditions, it takes around 20 ms to respond to a single request. [6] If the server does not have additional resources to fork a new thread, the request is queued and the server would postpone forking a new thread until a later time, when resources become available.

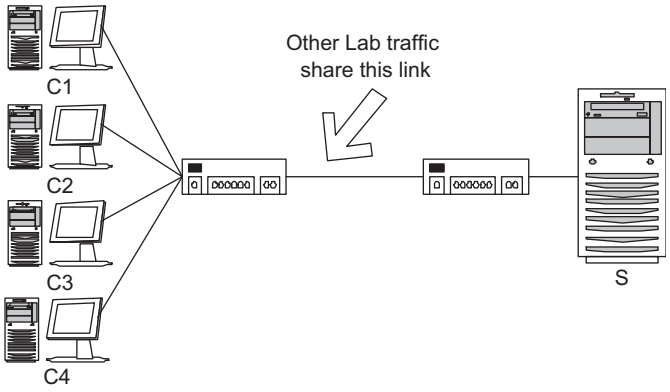


Fig. 5. Experimental setup used in our empirical evaluation.

3.4.2. RoQ exploit of the admission controller

Our first set of experiments demonstrates the impact of RoQ attacks on the PI admission controller outlined above. We have instantiated the target of the PI controller to be the memory utilization, measured as the ratio between used (physical) memory and total (physical) memory. When memory utilization is very high, frequent paging activities cause the system to thrash and as a result the service rate will dramatically decline. RoQ attacks then would push the system into thrashing, causing a decline in the admission ratio and hence a denial of service to legitimate requests for a long period of time until thrashing clears up. This process would repeat once the system recovers.

Fig. 6(left) illustrates the evolution of the admission ratio as a function of time. Under normal workload conditions (i.e., no overloading), the memory utilization is fairly low, and the admission ratio is about 1. At time 120, 800 requests are injected as a RoQ attack is initiated. As a result, the system is pushed into thrashing causing high memory utilization and associated paging activities. This inefficient period lasts for more than 200 s, until around time 500 when the admission controller recovers to admitting all legitimate requests. At time 740, another 800 requests are injected causing the same scenario to repeat. When K was chosen to be 0.01, the RoQ attacks achieved a potency of 8, i.e., a single attack request caused denial of service for eight legitimate requests.

Fig. 6(center) shows the effect of different K values on potency. As K gets larger, the potency decreases. Notice the resemblance in the trend between Fig. 6(left) and Fig. 4(left and centre) (obtained analytically), and also between Fig. 6(center) and Fig. 4(right) (also obtained analytically).

[5] In database end-systems, recovery could be very costly in terms of CPU cycles as well as I/O, due to rolling back of uncommitted transactions to insure database integrity.

[6] While accessing 1 Mbit of memory may seem a lot of memory for a simple web request, existing services such as back-end database servers and transaction web services could easily require more that figure of memory access to handle one request.

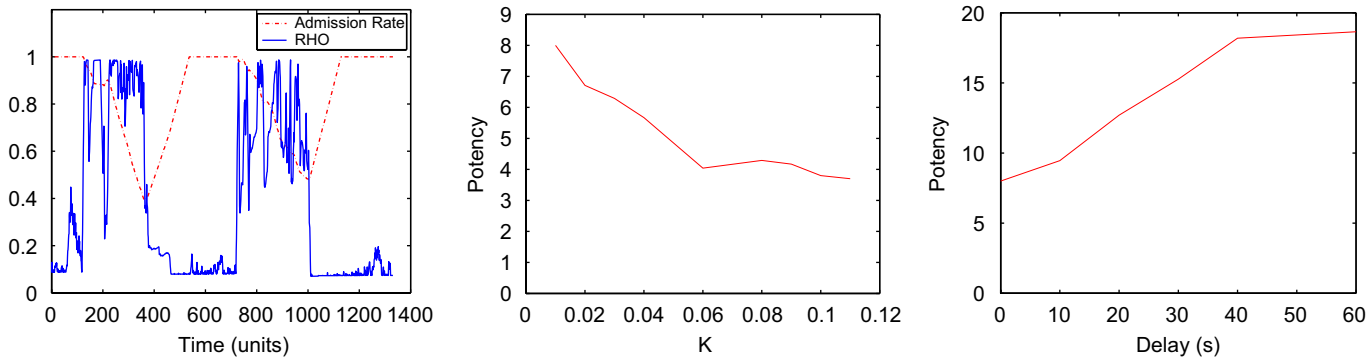


Fig. 6. Admission ratio and memory utilization under attack (left), effect of K on the potency (center) and effect of feedback delay on the potency (right).

3.4.3. Impact of feedback delay

In the experiments above, the effect of feedback delay was ignored. In these experiments, we assess RoQ exploits when the effect of feedback delay is taken into consideration. Feedback delay could arise from possibly several non-exclusive scenarios. First, feedback delay may arise from delay in measurements due to averaging for example. Methods like Exponential Weighted Moving Average (EWMA) for instance, tend to use a smoother value for control that is different from what is instantaneously experienced by the system. Second, the effect of feedback delay could be inherent in the design/architecture of the system. For instance, the measurement component could be located on a machine different from the one where control is applied. We modified our control rules to keep a short history of past utilization measurements, and applied the control rules on past values rather than the latest. Fig. 6(right) shows how the potency increases as feedback delay increases. When the feedback delay is around 60 s, the potency is higher than 18! Intuitively, long feedback delays have the effect of admitting more requests at the beginning since the admission controller does not know about the system state and whether it is thrashing. This confirms the known impact of feedback delay on the stability of control systems by causing them to become unstable, and in our case here, more vulnerable to RoQ exploits.

3.4.4. Admission control versus multiprogramming control

One of the main reasons making the admission controller in our experiments susceptible to RoQ exploits, is that the Mini-httpd server [34] blindly keeps forking threads as requests keep arriving, leaving the system at the mercy of its admission controller. Other web servers, such as Apache [4], maintain a *fixed* thread pool to prevent operating in an overload or a thrashing regime. A request is only processed when a pool thread is available. Effectively, this approach limits the level of concurrency (or multiprogramming level) in the system.

Indeed, having a fixed thread pool reduces the impact of RoQ attacks, but this comes at a cost—namely, there is a serious risk that the web server might become underutilized. Since requests could have very different characteristics, in terms of the resources they require, having a fixed pool of threads, and assuming the worst-case scenario, would prevent the web server from adapting to different mixtures of requests. In fact, this has prompted research studies into how the thread pool size may be adjusted dynamically based on the profiles of current (or recent) requests in the system [45,17].

Having a dynamically adjustable thread pool size is simply another paradigm for implementing admission control, and thus could itself be the target of RoQ attacks. Indeed, thread-pool-size adaptation could be exploited by forcing the thread pool size to oscillate between its two extremes (for example, by alternately subjecting the system to short requests, that do not take a lot of resources and long requests that would consume a lot of resources, which will cause the pool size to keep changing). Such tradeoff is very important to highlight. Do we want systems that are more resilient to attacks but less efficient, or systems that are efficient but susceptible to attacks?

3.4.5. Alternatives to admission control

In our analysis and experiments, we have assumed that upon the onset of thrashing, requests are "rejected". There are other alternative approaches that could be adopted in overload situations. For instance, content adaptation controllers are used to reduce the quality of the content serviced when the system is deemed to be operating under overload conditions. Another alternative would be to delay (or buffer) the requests until the system is operating efficiently. While such approaches may appear as possible defenses against RoQ exploits, they simply change the nature of damage.

For example, within the context of a content adaptation controller, an attacker's goal may well be to reduce the quality of the content returned to legitimate users. To that end, one can instantiate the model presented in Section 2 in such a way so as to capture the quality of the content as a function of the load on the system. Notice that since the degraded content is pre-computed and stored to avoid the overhead of real-time transcoding, the degradation would typically proceed in a few pre-defined discrete steps. A natural metric to capture the damage would be the difference in quality of the content before and after the attack. RoQ exploits of content adaptation controller will likely yield a high potency since it is hard for the system to figure out when to revert back to serving full content [1]. Reverting early would cause an instant overload situation.

Thus, it would typically take the system a long period of time to recover, during which, degraded content will be serviced. Another possible attacker's goal is to cause inconvenience (e.g., answering challenges) for legitimate users, with the hope of driving some of them away from using the service. This can be achieved by exploiting adaptation mechanisms that are used to differentiate between real users and zombie clients in overload situations [26].

The use of buffering as an approach for overload mitigation is not practical either, since for many applications, the added delays may introduce unintended consequences (e.g., triggering timeouts at other layers, which will effectively result in a DoS). Buffering (as a defense mechanism against RoQ exploits) simply changes the nature of damage—from a reduction in the capacity of the system to an increase in the response time of the system. In the next section, we study the impact of RoQ attacks that aim to increase the response time of the system; we do so by exploiting the adaptation dynamics in a load balancing setting.

4. RoQ attacks on load balancers

4.1. Adaptation through load balancing

Load balancers are integrated in the design of many end-systems to ensure efficient resource utilization and improved performance through load distribution. Generally speaking, load balancing mechanisms could be classified into two categories: static and dynamic. *Static load balancers* use a prescribed assignment strategy (such as round-robin or weighted round-robin) to make load distribution decisions [18,15]. *Dynamic load balancers*, on the other hand, rely on feedback from the resources they manage to inform their assignment decisions. In this section, we present a detailed model for studying the vulnerabilities of dynamic load balancers. We limit our exposition to load balancers employed in server farms. However, we believe that the model we present can be extended to load balancing solutions deployed in other settings, such as routers and network switches, routing protocols, firewalls and traffic shapers.

4.2. Model derivation

Consider the setup illustrated in Fig. 7, consisting of N identical servers and a load balancer. Requests arrive according to a Poisson process with an average rate of λ requests per second to the load balancer. All requests are assumed to be identical and each requires a fixed service time of T_s seconds.[7] The offered load, in terms of λ, should always be less than the total service rate for all servers—otherwise the servers would not be able to handle the request stream whether load balancing is used or not. The load balancer picks the server that is to handle an incoming request based on some load balancing policy.

[7] We relax this assumption in our experiments, where we use heavy-tailed distributions for the service time.

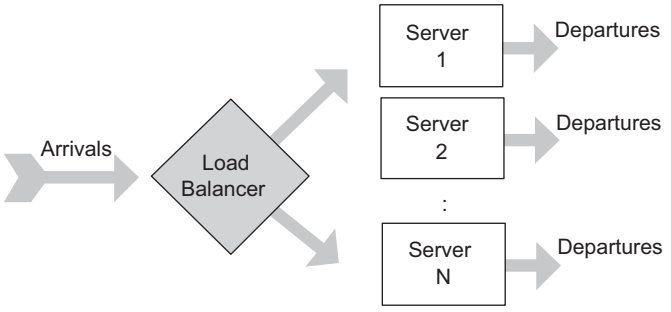


Fig. 7. The general setup we consider for load balancing composed of a load balancer and N identical servers.

Table 2
Parameters of the model used to analyze potency of RoQ exploits on proportional-control balancing policy

Parameter	Description
N	Number of servers
$\alpha^n(.)$	Admission ratio for server n
$q^n(.)$	Number of requests in queue for server n
λ	Rate of arrival of requests
T_s	Service time
β	Proportional-control balancing constant
T_q	Response time under no attack
$\acute{T}_q$	Response time under attack
$\bar{T}_q$	Response time under smoothed attack
D_{A}	Absolute damage
D_{S}	Sensitivity damage
μ	Service rate
δ	Attack rate
τ	Attack duration
T	Attack period

Under a static policy and assuming symmetry, the load balancer would assign $\frac{1}{N}$ of the arrivals to each server. Under dynamic load balancing, the load balancer would determine the admission ratio for each server based on the loads reported back from the servers.

Table 2 summarizes the notation and the description of the parameters used in our model.

Instantiating our general model of Section 2, the pricing function of the *load balancer* is given by the relationship between the response time of requests (prices) and the number of pending requests inside the server (load). The latter is function of both the admission ratio to that server and its service rate.

We consider the same RoQ attack pattern we considered in the previous section. However, in this setting, the attacker can distribute the attack magnitude M over one or more servers. It is possible for the attacker to bypass the load balancer, since load balancers in web-server setting adopt the "sticky connection" feature. This feature ensures that connections originating from the same client, over a period of time set by the "sticky timer", would always be directed to the same real server. While this feature is essential in maintaining sessions to track users, an attacker can now distribute its attack magnitude in any arbitrary manner across the servers, bypassing the load balancer.

Of course, if the servers have public IP addresses, the attacker does not need to leverage this feature as it is able to send the burst directly to the intended server(s). Throughout this section, we assume M would be distributed uniformly at random over the number of attacked servers.

A natural goal of a RoQ exploit on an load balancer is to maximize the damage caused by a periodic adversarial attack of magnitude M, where damage constitutes the *additional delay* that legitimate requests will experience. We instantiate two different metrics for computing the damage.

Absolute damage (D_A):

One can capture the damage resulting from this attack by the additional delay legitimate requests experience in comparison with their response time when the attack is not launched. In that case the absolute damage, D_A is given by

$$D_A = (\acute{T}_q - T_q) \times T\lambda, \tag{21}$$

where $\acute{T}_q$ is the average response experienced by legitimate requests under attack and T_q is the average response time under no attack.

Sensitivity damage (D_S):

One can also capture the damage resulting from this attack by the additional delay that legitimate requests experience, in comparison with another system where the attack burst would be smoothed over time according to another Poisson Process with a different arrival rate, through the load balancer. Thus this damage measures the sensitivity of the response time to the *burstiness* of the attack traffic, as opposed to the *magnitude* of the attack traffic. In particular, the damage, D_S is given by

$$D_S = (\acute{T}_q - \bar{T}_q) \times T\lambda, \tag{22}$$

where $\bar{T}_q$ is the average response time these requests would have experienced, if the attack burst was smoothed over time. Clearly, the absolute damage is larger than the sensitivity damage, since T_q is always less that $\bar{T}_q$ due to the absence of attack traffic. [8]

We now turn our attention to calculating the cost (to the attacker) for mounting the attack. Such cost can be captured by several metrics, including the attack magnitude, M, or the attack rate, $\frac{M}{T}$. We chose to define the cost as the time spent by the attack requests in service. This metrics reflects the capacity (or "energy") used by the system to process the adversarial workload. In particular, the cost, C is given by

$$C = M \times T_s. \tag{23}$$

Using the expressions for damage (D) and cost (C), one can compute the attack potency using the definition in equation (1), where D could be instantiated as the absolute damage or the sensitivity damage (D_A or D_S).

[8] Using M/D/1 queuing model, we calculate T_q, $\acute{T}_q$ and $\bar{T}_q$ based on the attack parameters [22].

Notice that our choice of both metrics, damage and cost, as units of time enables us to have a unit-less metric (the potency) which describes the trade-offs in time. For example, a potency of 100, would mean that for every second, the attack requests spend in service, 100 s of additional delay get added to the response time for the legitimate requests.

Consider a dynamic discrete-time model, where servers relay their load to the load balancer in order to influence future balancing decisions. Let $q^n(i)$ denote the queue size, at time i, for server n. The queue size evolves according to the following equations:

$$q^n(i) = q^n(i-1) + \alpha^n(i)\lambda - \mu^n, \tag{24}$$

where $\alpha^n(i)$ is the percentage of requests admitted to server n—a percentage that is set by the load balancer. μ^n is the service rate for server n. [9] Clearly, $\sum_{n=1}^{N} \alpha^n(i)$ at any time instant, i, is 1. Eq. (24) is bounded from below by 0. [10]

Ideally, servers should measure their load and report that back to the load balancer. We use the queue size as a load metric—the larger the queue size, the more loaded the server is. We relax this assumption in our experimental evaluation since we allow requests to be of different sizes.

Periodically, servers will relay their loads to the load-balancer. The load balancer would then adjust the admission ratio for each server based on the load-balancing policy. We have studied few of the most-commonly used mechanisms [18,15] in a recent work of ours [22]. We present only the proportional-control balancing policy here.

Ideally, the load balancer should match the queue sizes. A simple controller to achieve such a goal can be described by the following proportional controller:

$$\alpha^n(i) = \frac{1}{N} + \beta \sum_{j=1}^{N} (q^j(i-1) - q^n(i-1)), \tag{25}$$

where $\alpha^n(i)$ is adjusted based on the differences between queue sizes. β, a key parameter in the controller, is introduced to smooth out the difference for stability reasons. Notice that instantaneous queue size will likely exhibit oscillations that would prevent it from being used directly as an accurate measure of load, unless reported instantly to the load balancer. [11] However, due to the feedback delay coupled with the overhead of communicating instantaneous queue sizes to the load balancer, a more smoothed signal is used. One can easily see that when all servers have the same queue size, $\alpha^n(i)$ will be $\frac{1}{N}$. Also, at any time instant i, $\sum_{j=1}^{N} \alpha^j(i)$ will be 1.

[9] Throughout this paper, we assume all N servers have the same service rate μ, thus we occasionally drop the superscript.

[10] In our analysis, we assume infinite queue size and hence no bounds from above.

[11] Indeed, if instantaneous load measures are reported instantly to the load balancer, the load balancer could perfectly adapt to noisy/short-term changes. Communicating instantaneous measures may not be feasible in practice.

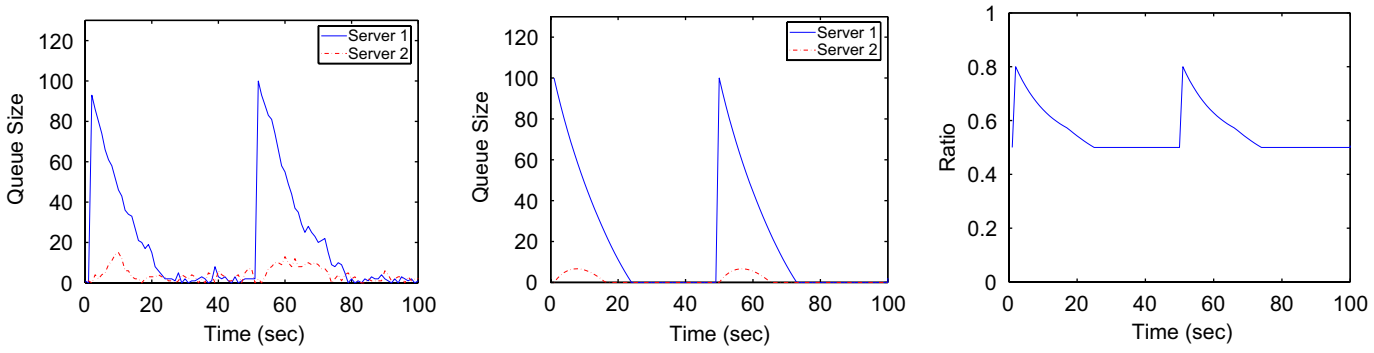


Fig. 8. Vulnerability of load balancing under proportional control: simulation results (left) and numerical solutions (center) and (right).

4.3. Numerical solution

We now solve the above difference equations numerically to capture the effect of the attack traffic. For simplicity, we ignore the stochastic effect of the request arrival process and present results under a fluid model.

In a smooth fluid model, as long as the arrival rate is less than the service rate, there will be no queuing. However, if the arrival rate exceeds the service rate, queuing would occur. We will also validate our model with simulation results, which will show that despite the limitations of the fluid model, the numerical solutions we derive still capture the essential dynamics involved, enabling us to accurately assess the damage inflicted by an adversary.

Notice that due to the absence of stochastic effects, we cannot differentiate between the case when the attack traffic is not present versus when the attack traffic is smoothed, since in both cases the queue size would be zero as the arrival rate never exceeds the service rate. Hence, we give only the damage as the increase in response time observed by legitimate requests compared to observing an empty queue (of size 0).

We take N equal to 2, and we assume that requests arrive with a rate $\lambda = 15$ requests per second, that the service time T_s is fixed to 0.1 s, and that the attack burst of magnitude 100 arrives to one of the servers and is repeated every attack period of 50 s.

Fig. 8 illustrates the results we got from using a *proportional-control balancing* policy. We have chosen β to be 0.003. Fig. 8(left) show the queue sizes for both servers as the attack is launched on server 1, results obtained from a simulation experiment under the same parameters. Fig. 8(center) show the queue sizes obtained from the numerical solution. Notice that the results match pretty well. Fig. 8(right) shows the admission ratio for server 2 obtained from the numerical solution.

Unlike the static balancing policy which would have affected only one server, the attack had an impact on both servers, even though only one of them was targeted by the adversary. This is the result of the dynamic load balancer's diversion of subsequent requests to the other server, which temporary exceeds its service rate causing queuing. Notice that the attacked server was able to get rid of the burst and return to its normal operation at around time 22 s (as opposed to 40 s, in the static case with same parameters). However, for the other server, the situation is different. It has to deal with the extra load being diverted to it. That is why we see an increase in its queue size before it can also get back to its normal operation. The above attack resulted in an attack potency of 65 using the absolute damage metric and 56 using the sensitivity damage metric. These results are computed in simulation experiments averaged over 10 independent runs. The potency we got from the numerical solution was pretty consistent at 73.

Fig. 9(left) shows simulation results obtained from a setup with 6 servers. We vary the number of servers attacked (on the x-axis) from 1 to 6 and we plot the absolute potency on the y-axis, for different β values. Each point is averaged over 10 independent runs. One can see the impact of the parameter β on the potency. Notice also that attacking one server is the best attack strategy under static load balancing. We also plot the optimal load balancing policy (which is simply making instantaneous decisions based on clairvoyant knowledge of attack parameters). While such "optimal" load balancer is impossible to implement in practice, the presentation of these results is useful as they show a lower bound on the achievable potencies. Distributing the attack traffic over all servers, is the best attack strategy (highest potency) under optimal balancing. In general, the optimal number of servers to attack is a function of the β parameter.

Fig. 9(right) shows the impact of a 10-s feedback delay on the absolute potency under the proportional load balancer with different values of β. Notice that the large β values that decreased the potency close to the optimal when no feedback delay was present (Fig. 9(left)), are now the ones that cause the maximum potency ($\beta = 0.0015$ and 0.0009). That is because the load balancer is reacting in the wrong manner (aggressively) and at the wrong time (too late). In these cases, it is better not to do dynamic load balancing, i.e., switch to static load balancing instead. If β is small, however, the load balancer reacts slowly and becomes less sensitive to the feedback delay.

4.4. Internet experiments

To validate the results we obtained, we have experimented with the proportional-control load balancing policy. Additional results could be found here [22].

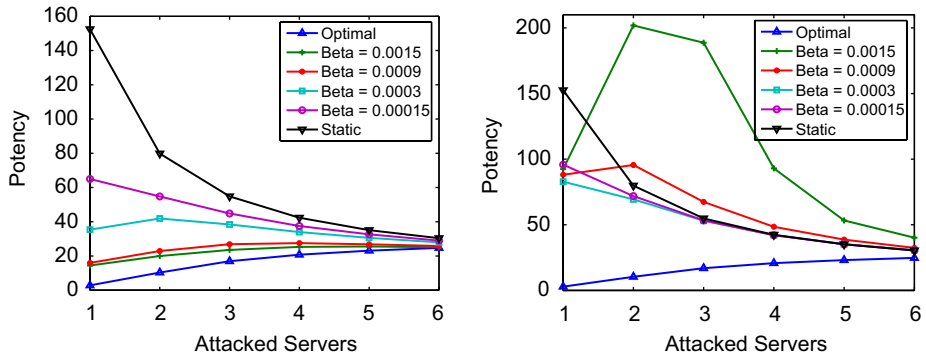


Fig. 9. Vulnerability assessment for proportional load balancer in comparison to the static and the optimal load balancing policies: without feedback delay (left) and with feedback delay of 10 s (right).

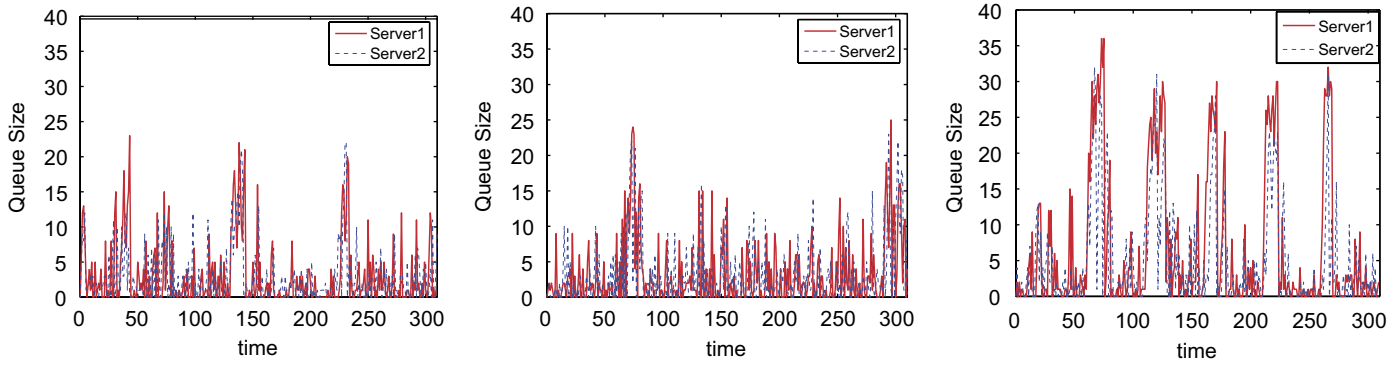


Fig. 10. Performance of a proportional balancer ($\beta = 0.003$): no attack (left), attack traffic is smoothed (center), and under attack (right).

4.4.1. Experimental setup

Fig. 7 depicts the experimental setup we used in running our experiments. It consists of a machine running the load balancer, and two machines running MINIHTTPD [34], and several client machines generating web traffic using HTTPERF [36]. All machines are of 1 Ghz/256 MB AMD Athlon(tm) and running Linux 2.4.20. We modified MINIHTTPD and HTTPERF to communicate with the load balancer, which is responsible for selecting a server for each client request. For each connection request, the load balancer will select a MINIHTTPD server according to the load balancing policy. As a result, the client initiating the request will establish a normal HTTP connection with the selected MINIHTTPD server. Feedback information is sent by every MINIHTTPD server periodically to the load balancer. Linux does not provide any system calls for applications to get the listen queue size. Thus, we use the number of active requests (accepted connections) as an approximation of the total number of pending requests, which constitute the feedback signal to the load balancer. This is similar to what most software monitoring solutions report to load balancers in practice. Since MINIHTTPD will fork a new thread for each new accepted connection, the queue size is also the number of currently running threads that deal with the requests in the system. The multithreaded nature of MINIHTTPD implies that multiple requests can be handled by different threads in a round-robin manner (as opposed to a pure queuing model).

In our experiments, we set the number of servers to two. Each experiment lasts for 310 s. The attack requests are sent to one of the server bypassing the load balancer. As mentioned before, this can be achieved in practice by using the sticky connection feature found in most load balancers. The request service time follows a Pareto distribution to reflect the highly variable nature of HTTP connection time—e.g., due to the heavy-tailed nature of file size distributions [5], for example. The parameters of the Pareto service time distribution was set to (2,0.05) with an upper bound of 250 s and a mean of 100 ms. Request arrivals follow a Poisson process with a mean rate of 15 requests/s. The attack workload was chosen to consist of 100 attack requests that are injected every 50 s for five times. All experiments are allowed to warm-up for 60 s before measurements were collected. Fig. 10 shows the queue size for both servers under no attack (left), when attack traffic is smoothed (center) and under attack (right). The above experiments resulted in an attack potency of more than 50 using the absolute damage metric and more than 40 using the sensitivity damage metric. Additional experiments could be found here [22].

5. RoQ implications and defense mechanisms

5.1. Measurement-based on-line tuning of RoQ attacks

The knowledge of the exact recovery time enables an attacker mounting RoQ exploits to optimize its attack parameters so that bursts would arrive in the correct time. If the attacker under estimates the exact recovery time, two bursts would be close to each other and would result in an increase in the overall "cost" in mounting the attack. Similarly, if the attacker over estimates the exact recovery time, two bursts would be far from each other and would decrease the overall "damage". In both cases, the attacker would still cause harm, but only with a lower potency value. In some settings, the knowledge of the system recovery time is easy to obtain. For example, an attacker mounting a RoQ exploit on a content adaptation controller can determine exactly when the system has recovered, by observing the quality of the content serviced. In other settings, this could be more challenging. A promising approach to correctly time the attack traffic is to use measurements to probe the state of the system. Consider RoQ exploits on admission controllers; it is reasonable to assume that the attacker can send few probing requests over time and following an attack burst. Based on the number of these probes that got admitted, an attacker can estimate the admission ratio (perhaps with some measurement error). This will enable the attacker to effectively figure out whether the admission controller has recovered or not and more importantly when is the best time to repeat its attack. A similar approach would be used in a load balancing setting, based on the response time of the attacker's own requests. Once requests start taking the minimum response time, the attacker can tell that the load balancer has recovered. Notice that such measurement-based on-line tuning of RoQ attacks could render some defenses ineffective, if such defenses rely on figuring out the attack periodicity, for example.

5.2. Detection and trace-back

An adversary mounting a RoQ attack does not have to overwhelm the system constantly in order for its attack to be effective. Moreover, the transients induced by the attack are not much different from those that are possible under normal operation (except that they do not subside). These dimensions of RoQ attacks make it challenging for an end-system to even realize that it is under attack. Notice that tracing back the perpetrators and taking counter measures is challenging since the attacker could be launching its attack through zombie clients, with different IP addresses. This adds to the complexity of detection and trace-back.

5.3. Possible defense mechanisms

From our analysis and experimental evaluation of the various exploits of system dynamics, it is evident that tuning control parameters—such as K in the admission controller and β in the proportional control load balancer—results in different system behaviors. In particular, these parameters expose the tradeoffs between efficiency and tolerance to RoQ exploits. Tuning them to achieve the best performance may lead to settings that would make the system quite vulnerable to RoQ exploits. On the other hand, selecting parameters that may minimize the damage from RoQ exploits may lead to very inefficient (or sluggish) "normal" operation. This suggest that it could be advantageous for these parameters to be adjusted on-line, based on whether or not the system is suspected to be under a RoQ exploit. The goal of a defense mechanism is to mitigate the effect of the attack without compromising the performance when no attacks are present.

To that end, we give a flavor of such possible defense mechanism based on some simulation results, in which we adapted in an on-line fashion the parameter β for the proportional-control load balancer.

For load balancers, Fig. 11(left) shows the admission ratio for one of the two servers in our experiments when β was set to 0.003. Fig. 11(center) shows the admission ratio for one of the two servers when β was set to 0.03. Fig. 11(right) shows the admission ratio for one of the two servers when β was allowed to dynamically adapt. One can easily see that such adaptation has the benefit of reacting fast to sudden changes (during attack)

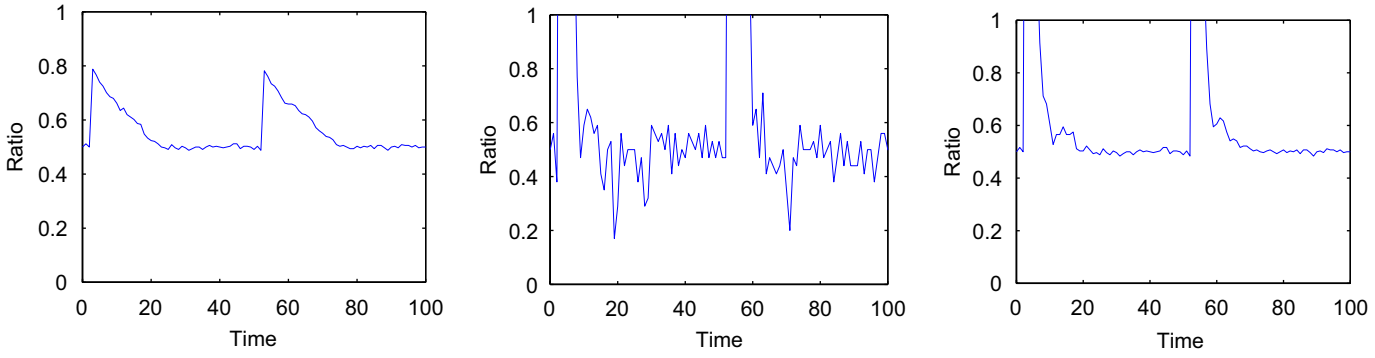


Fig. 11. Simulation results for the admission ratio for a single server under a proportional balancing policy: β fixed at 0.003 (left), β fixed at 0.03 (center), and β is adapted over time to mitigate the impact of the attack (right).

and also of having a smooth operation when changes are not significant (legitimate burstiness).

Similarly, for admission controllers, the parameter K (the gain of the admission controller) could be adjusted in an online fashion, so as to react faster when there are sudden changes and slower when the changes are not significant.

6. Related work

The work presented in this paper relates to a fairly large body of literature. We briefly exemplify the different dimensions of this body of work below.

6.1. End-system adaptation mechanisms

End-systems employ different forms of adaptation mechanisms. (1) Admission control strategies are employed in operating systems (by suspending or terminating processes) to ensure that virtual memory performance is not compromised as a result of excessive swapping [47,45]. They are employed in real-time and multimedia systems to ensure that the QoS of admitted tasks is not compromised when additional tasks are admitted into the system [27,11,14,41]. They are employed in web/media server designs to ensure that a maximum response time is not exceeded [25,42,12,13,43]. (2) Content adaptation mechanisms are employed to mitigate the overload conditions by reducing the quality/quantity of the content serviced. They are employed in a wide range of setups; from web content adaptation [1,39] to multimedia service rate adjustments [44], among others. (3) Load balancers are integrated in the design of most scalable and distributed applications and services. Typically, they are embedded as part of the infrastructure supporting these applications and services—e.g., as part of routers and network switches [15,16], routing protocols [19], firewall and traffic shapers [37,18], HTTP and database server farms [31–33,24], among others. These techniques, however, did not investigate the adversarial exploitation of the adaptation dynamics for the purpose of reducing one or more aspects of service quality, or of efficiency. Rather, they focused mostly on tuning the adaptation mechanism to ensure the quiescent operation of the end-system behind it.

6.2. Control-theoretic modeling and analysis

Marshaling techniques from control and optimization theory has been a fruitful direction as evidenced by the works in [3,40,2,7,29,17]. In that respect, we single out the works in [3,40], which investigated the use of a PI controller to adjust the admission ratio for an Apache Web server in order to operate in a stable manner. In [2], a feedback control loop was incorporated in an Apache web server in order to adjust the relative delays for different classes through dynamic scheduling. In [29], Q-PID, a new admission control mechanism, was introduced. The idea is to adjust the admission ratio in order to guarantee a bounded response time for the users. In [7], nonlinear optimization theory was used to optimize the performance of web servers through breaking sessions into stages and performing admission control with an eye on maximizing an application-specific reward function. Again, these studies did not focus on the adversarial aspect we considered in this paper, but rather on controlling and optimizing the web server behavior. Indeed, they did not even recognize or consider the adaptation strategies that they advocated as potential vulnerabilities worthy of characterization.

6.3. RoQ versus other attacks

DoS attacks [10,8] and its many variants [9] could be characterized as targeting one dimension of a system's service quality—namely, its availability. There are a number of papers that classify various forms of DoS attacks; examples include [23,35,30]. Using our model, DoS attacks could be classified as RoQ attacks with an infinite aggressiveness index (defined in Section 2), which imply that the attacker's ultimate goal is to maximize the damage at any cost. In this paper, we have focused on attacks whose perpetrators are not focused on denying access (i.e., targeting availability), but rather they are focused on bleeding the system of its capacity, or simply pushing it to operate in inefficient operating regions to reduce some aspect of service quality. More importantly, in this paper, we have focused on the harder-to-detect, low-intensity attacks, i.e., with modest aggressiveness compared to the aggressiveness required for DoS attacks. On the other hand, the "shrew" attack proposed in [28] is an example of a low-intensity, harder to detect attack which targets a set of flows to cause them to timeout. Clearly, the scope of shrew attacks is limited to targeting TCP flows which employ the timeout mechanism. RoQ attacks have much broader scope for targeting adaptation mechanisms that can be found in modern computing systems.

7. Conclusion

In this work, we exposed variants of RoQ attacks that target end-systems through exploiting the transients of their adaptation mechanisms. RoQ attacks are identified as those attempting to maximize the marginal utility of the attacker's workload (attack "Potency"). We used a control-theoretic framework to underline the complex interplay between the efficiency–load behavior of a resource and the adaptation mechanisms of both the resource and its consumers. We have instantiated this framework, using more detailed models, on admission controllers and load balancers. We have shown that RoQ attacks can introduce significant inefficiencies or can increase the response time of legitimate requests, while evading detection by consuming a small portion of the hijacked capacity over long-time scales. We confirmed our findings through real Internet experiments performed in our lab. We believe that it is very important to develop a general understanding of the design principles that could be adopted to protect against RoQ exploits. In particular, the tradeoff between performance under normal operation and resiliency against RoQ exploits is important to highlight.

References

[1] T.F. Abdelzaher, N. Bhatti, Web content adaptation to improve server overload behavior, Comput. Networks 31 (11–16) (1999) 1563–1577.

[2] T. Abdelzaher, C. Lu, Modeling and performance control of internet servers, in: Proceedings of the 39th IEEE Conference on Decision and Control (ICDC), Sydney, Australia, 2000.

[3] M. Andersson, M. Kihl, A. Robertsson, Modelling and design of admission control mechanisms for Web servers using non-linear control theory, in: Proceedings of ITCom, 2003.

[4] Apache HTTP Server, ⟨http://httpd.apache.org⟩.

[5] M. Balter, M. Crovella, C. Murta, On choosing a task assignment policy for a distributed server system, J. Parallel Distrib. Comput. 59 (2) (1999) 204–228.

[6] A. Bestavros, N. Katagai, J. Londono, Admission control and scheduling for high performance World Wide Web servers, Technical Report BUCS-TR-1997-015, Boston University, Computer Science Department (August 1997).

[7] J. Carlstrom, R. Rom, Application-aware admission control and scheduling in web servers, in: Proceedings of Infocom, 2002.

[8] C.C. Center, CERT Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks, ⟨http://www.cert.org/advisories/CA-1996-21.html⟩, Original issue date: September 19, 1996.

[9] C.C. Center, Trends in Denial of Service Attack Technology—October 2001, ⟨http://www.cert.org/archive/pdf/DoS_trends.pdf⟩.

[10] C.C. Center, Denial of Service Attacks, ⟨http://www.cert.org/tech_tips/denial_of_service.html⟩.

[11] S. Chatterjee, J.K. Strosnider, A generalized admissions control strategy for heterogeneous, distributed multimedia systems, ACM Multimedia (1995) 345–356.

[12] L. Cherkasova, P. Phaal, Session based admission control: a mechanism for improving the performance of an overloaded Web server, Technical Report HPL-98-119, HP Labs (June 1998).

[13] L. Cherkasova, P. Phaal, Predictive admission control strategy for overloaded commercial Web server, in: Proceedings of Eighth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2000.

[14] T. Chiueh, M. Vernick, An empirical study of admission control strategies in video servers, in: Proceedings of the 1998 International Conference on Parallel Processing, Minneapolis, MN, 1998, pp. 313–320.

[15] Cisco, Configuring Load Balancing on the CSS 11500, ⟨http://www.cisco.com/warp/public/117/methods_load_bal.pdf⟩.

[16] Cisco, Network Implementation, ⟨http://www.cisco.com/univercd/cc/td/doc/product/iaabu/localdir/ld20rns/ldicgd/ld3_ch3.pdf⟩.

[17] Y. Diao, N. Gandhi, S. Parekh, J. Hellerstein, D. Tilbury, Using mimo feedback control to enforce policies for interrelated metrics with application to the apache web server, in: Proceedings of the Network Operations and Management Symposium 2002, Florence, Italy, 2002.

[18] F5, BIG-IP Load Balancer Limited, ⟨http://www.f5.com/f5products/products/bigip/ltm/lbl.html⟩.

[19] B. Fortz, M. Thorup, Internet traffic engineering by optimizing OSPF weights, in: Proceedings of IEEE INFOCOM, 2000.

[20] M. Guirguis, A. Bestavros, I. Matta, Exploiting the transients of adaptation for RoQ attacks on Internet resources, in: Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP), 2004.

[21] M. Guirguis, A. Bestavros, I. Matta, Y. Zhang, Reduction of quality (RoQ) attacks on Internet end systems, in: Proceedings of INFOCOM, 2005.

[22] M. Guirguis, A. Bestavros, I. Matta, Y. Zhang, Adversarial exploits of load balancers: vulnerability assessment and design tradeoffs, in: Proceedings of INFOCOM, May, 2005.

[23] A. Hussain, J. Heidemann, C. Papadopoulos, A framework for classifying denial of service attacks, in: Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, ACM Press, New York, 2003, pp. 99–110.

[24] IBM, DB2 connection routing using Linux load balancing, ⟨http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0410wright/⟩.

[25] X. Jiang, P. Mohapatra, An aggressive admission control algorithms for multimedia servers, in: Proceedings of International Conference on Multimedia Computing and Systems, 1997, pp. 620–621.

[26] S. Kandula, D. Katabi, M. Jacob, A. Berger, Botz-4-Sale: surviving organized DDoS attacks that mimic flash crowds, in: Proceedings of NSDI, Boston, MA, 2005.

[27] E. Knightly, N. Shroff, Admission control for statistical QoS: theory and practice, IEEE Network 13 (2) (1999) 20–29.

[28] A. Kuzmanovic, E. Knightly, Low-rate TCP-targeted denial of service attacks (The Shrew vs. the Mice and Elephants), in: Proceedings of ACM SIGCOMM'03, karlsruhe, Germany, 2003.

[29] S. Lim, C. Lee, C. Ahn, C. Lee, K. Park, An adaptive admission control mechanism for a cluster-based web server system, in: Proceedings of International Parallel and Distributed Processing Symposium (IPDPS), Fort Lauderdale, FL, 2002.

[30] C. Meadows, A formal framework and evaluation method for network denial of service, in: Proceedings of the 12th IEEE Computer Security Foundations Workshop, 1999.

[31] Microsoft, SharePoint Services, ⟨http://www.microsoft.com/resources/documentation/wss/2/all/adminguide/en-us/stsf15.mspx⟩.

[32] Microsoft, Network Load Balancing Technical Overview, ⟨http://www.microsoft.com/technet/prodtechnol/windows2000serv/deploy/confeat/nlbovw.mspx⟩.

[33] Microsoft, SQL Server 2000 High Availability Series: Implementing Network Load Balancing, ⟨http://www.microsoft.com/technet/prodtechnol/sql/2000/deploy/hasog04.mspx⟩.

[34] mini_httpd: small HTTP server, ⟨http://www.acme.com/software/mini_httpd⟩.

[35] J. Mirkovic, J. Martin, P. Reiher, A taxonomy of DDoS attacks and DDoS defense mechanisms, Technical Report 020018, Computer Science Department, University of California, Los Angeles.

[36] D. Mosberger, T. Jin, Httperf: a tool for measuring web server performance, in: Proceedings of the First Workshop on Internet Server Performance (WISP '98), Madison, WI, 1998.

[37] Nortel, Alteon Web OS Traffic Control, ⟨http://www.nortelnetworks.com/products/01/webos/index.html⟩.

[38] K. Ogata, Modern Control Engineering, fourth ed., Prentice-Hall, Englewood Cliffs, NJ, 2002.

[39] R. Pradhan, M. Claypool, Adaptive content delivery for scalable Web servers, in: Proceedings of the International Network Conference (INC), Plymouth, United Kingdom, 2002.

[40] A. Robertsson, B. Wittenmark, M. Kihl, Analysis and design of admission control systems in Web-server systems, in: Proceedings of American Control Conference (ACC), 2003.

[41] S. Son, K. Kang, QoS Management in Web-based real-time data services, in: Proceedings of the Fourth IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS'02), Newport Beach, California, 2002.

[42] T. Voigt, Overload behaviour and protection of event-driven Web servers, in: Proceedings of International Workshop on Web Engineering (in conjunction with Networking 2002), Pisa, Italy, 2002.

[43] T. Voigt, P. Gunningberg, Handling multiple bottlenecks in Web servers using adaptive inbound controls, in: Proceedings of Protocols for High-Speed Networks, 2002, pp. 50–68.

[44] X. Wang, H. Schulzrinne, Adaptive reservation: a new framework for multimedia adaptation, in: Proceedings of the IEEE International Conference on Multimedia and Expo (ICME), New York, NY, 2000.

[45] M. Welsh, D. Culler, Overload management as a fundamental service design primitive, in: Proceedings of the Tenth ACM SIGOPS European Workshop, Saint-Emilion, France, 2002.

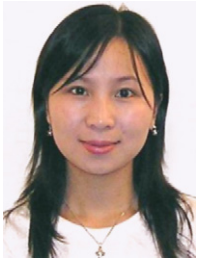
[46] M. Welsh, D. Culler, Adaptive Overload Control for Busy Internet Servers, in: Proceedings of the 4th USENIX Conference on Internet Technologies and Systems (USITS), 2003.

[47] M. Welsh, D.E. Culler, E.A. Brewer, SEDA: an architecture for well-conditioned, scalable Internet services, in: Symposium on Operating Systems Principles, 2001, pp. 230–243.

Mina Guirguis is assistant professor of computer science at Texas State University. He completed his Ph.D. in computer science at Boston University in 2006. He received his Bachelor degree in computer science and automatic control from Alexandria University in 1999 and his MA degree in computer science from Boston University in 2005. His research interests include security aspects in computing systems and networks, sensor networks, overlays and P2P networks.

Azer Bestavros obtained Ph.D. in Computer Science from Harvard University in 1992. He is Professor and Chairman of Computer Science at Boston University. His research interests are in the general areas of networking and real-time systems. His seminal works include his probabilistic generalization of classical rate-monotonic analysis, his pioneering of the push model for Internet content distribution adopted years later by CDNs, and his characterization of Web traffic self-similarity and reference locality. His research work has culminated so far in 10 Ph.D. theses, over 80 masters and undergraduate student projects, and two startup companies. With over 3000 citations to his papers, CiteSeer ranks him in the top 5% of its list of 10,000 most-cited authors in Computer Science. His research has been funded by government and industry grants totaling over $15M. He received distinguished service awards from both the ACM and the IEEE, and is a distinguished speaker of the IEEE.

Ibrahim Matta received his Ph.D. in computer science from the University of Maryland at College Park in 1995. He is an associate professor of computer science at Boston University. His research involves routing and transport protocols, focusing on resiliency and safety aspects. He published over 70 refereed technical papers, and was guest co-editor of three special journal issues. He received the National Science Foundation CAREER award in 1997. He is on the Editorial Board of the Computer Networks Journal. He was General Chair of WiOpt'06, Technical Program Co-chair of ICNP'05, Technical Program Co-chair of SenMetrics'05, Internet Co-chair of Infocom'05, Publication Chair of Infocom'03, and Tutorial and Panel Chair of Hot Interconnects'01. He was co-organizer and Technical Program Co-chair of the EU-US NeXtworking'03.



Yuting Zhang completed her Ph.D. in Computer Science at Boston University in 2006. She received her Bachelor and Master degrees in computer science from the University of Science and Technology in Beijing. During the summer of 2005 and the spring of 2006, she worked at VMware as an intern in the resource management group. Currently, she is assistant professor of computer science at Allegheny College. Her research interests are in the general areas of operating systems and networks. Her research includes scheduling in multimedia and embedded real-time systems, resource management in virtual execution environments, and network security. She has authored or co-authored about 10 papers published in different conferences and journals such as RTSS, INFOCOMM, VEE, RTCSA, TOC. She has also served as a reviewer for RTAS'06, RTSS'05, ECRTS'05, RTSS'04 and ISCC'04.