Reduction of Quality (RoQ) Attacks on Internet End-Systems †

MINA GUIRGUIS AZER BESTAVROS IBRAHIM MATTA YUTING ZHANG
{msg, best, matta, danazh}@cs.bu.edu
Computer Science Department
Boston University
Boston, MA 02215, USA

***Abstract*— Current computing systems depend on adaptation mechanisms to ensure that they remain in quiescent operating regions. These regions are often defined using efficiency, fairness, and stability properties. To that end, traditional research works in scalable server architectures and protocols have focused on promoting these properties by proposing even more sophisticated adaptation mechanisms, without the proper attention to security implications. In this paper, we exemplify such security implications by exposing the vulnerabilities of admission control mechanisms that are widely deployed in Internet end systems to Reduction of Quality (RoQ) attacks. RoQ attacks target the transients of a system's adaptive behavior as opposed to its limited steady-state capacity. We show that a well orchestrated RoQ attack on an end-system admission control policy could introduce significant inefficiencies that could potentially deprive an Internet end-system from much of its capacity, or significantly reduce its service quality, while evading detection by consuming an unsuspicious, small fraction of that system's hijacked capacity. We develop a control theoretic model for assessing the impact of RoQ attacks on an end-system's admission controller. We quantify the damage inflicted by an attacker through deriving appropriate metrics. We validate our findings through real Internet experiments performed in our lab.**

***Index Terms*— Security; Denial of Service; Scalable Web Services; Adaptive Resource Management; Performance Evaluation.**

I. INTRODUCTION

End-system computing systems 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 adaption mechanisms increases, it becomes harder to understand their dynamic behavior. Specifically, they 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, our 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 electric and mechanical systems, are evaluated. For such systems, the characterization of system dynamics is front and center to protect against oscillatory behaviors and instabilities.

Traditionally, system dynamics are "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, system 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. 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 [1] 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 focus on RoQ exploits of adaptation strategies that are widely deployed in Internet end-systems.

An Illustrative RoQ Exploit: Current adversarial strategies for Denial of Service (DoS) are *brute force* [2]. An attacker may render a system, say a Web server, useless by subjecting that system to a sustained attack workload (*e.g.,* syn attack)

† This work was supported in part by NSF grants ANI-0095988, ANI-9986397, EIA-0202067 and ITR ANI-0205294.

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 [3], [4], [5]. 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.

Paper Outline: Section II 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 an persistent attempt to regularly knock the system off its stable (or quiescent) operating point. In Section III, we present a more detailed analytical model for an admission controller employed in an end-system web server. We derive closed formulas to quantify the damage inflicted by an attacker. In Section IV, we present results from Web server experiments we have conducted, which confirm the feasibility of RoQ attacks and provide further validation of the insights we gained from analysis and simulations. In Section V, we briefly discuss related work, noting that throughout this paper, we point to various pieces of research work as appropriate. We conclude in Section VI with a summary and with future directions.

II. RoQ Attack Premise and Definition

This paper leverages some of the models and analysis of RoQ attacks presented in earlier work of ours [1]. 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.

A. 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.

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

The Potency definition given by equation 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, 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 equation 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. Throughout this paper, we take Ω to be 1.

B. 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/sec). 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

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

function, called the system's Lyapunov function [6]. 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 equation 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.

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

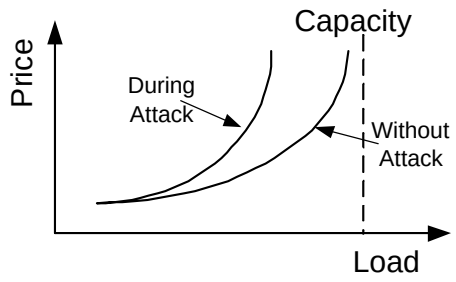

Fig. 1. An example of a web server pricing function as the load on the system varies. RoQ attacks, effectively, will keep the pricing function changing.

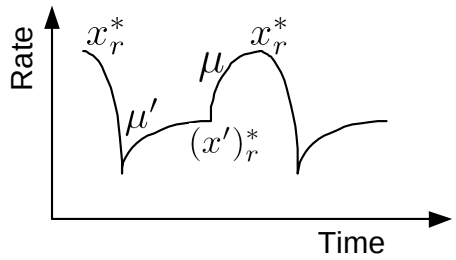

Fig. 2. Effect of a RoQ attack pattern on the request rate ($x_r(t)$). RoQ attacks will hinder convergence to steady state points.

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 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 Figure 1. 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 Figure 1. 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 Figure 2, 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 Equation 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 equation 4, where Ω reflects the relative values that an attacker attributes to "damage" versus "cost", or equivalently the desired level of aggression.

$$\begin{aligned} \Pi &= \frac{\delta(\frac{1}{\mu'}+\frac{1}{\mu})}{\left(\delta/(\frac{1}{\mu'}+\frac{1}{\mu})\right)^{1/\Omega}} \\ &= \delta^{1-\frac{1}{\Omega}}(\frac{1}{\mu'}+\frac{1}{\mu})^{1+\frac{1}{\Omega}} \end{aligned} \tag{4}$$

In the next section, we will consider a more elaborate analytical model to gain further insights into more complicated adaptation dynamics of specific systems that we could not cast in the generic optimization process we relied upon in this section.

III. RoQ Attacks on End-Hosts

Adaptation Through Admission Control: Admission controllers—a common fixture of computing systems and networks—are used to protect against overload conditions by rejecting (or postponing) 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 I illustrated, admission controllers

may be targets of RoQ exploits. In this section, we instantiate from the general model we presented in Section II, a detailed model for studying the vulnerabilities of admission controllers.

A. 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*. In this paper, we use a PI controller [6] 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, 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 Figure 3.

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

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

TABLE I
PARAMETERS OF THE LINEARIZED MODEL USED TO ANALYZE POTENCY OF ROQ EXPLOITS OF PI ADMISSION CONTROL.

Instantiating our general model of Section II, the pricing function of the *admission controller* is given by the relationship between the admission ratio of web requests $\alpha(.)$ and the utilization of the server $\rho(.)$. 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. Figure 4 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 II, 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, equation 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.

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

Equation 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]

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

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.

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 equation 7 for the evolution of the number of pending requests $n(.)$

$$\begin{aligned} n(i) \quad = \quad & n(i-1) + m(i) \\ & -(\mu_{\max} - I(\rho(i) > \rho_o) \times \omega(\rho(i) - \rho_o)) \end{aligned} \tag{7}$$

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

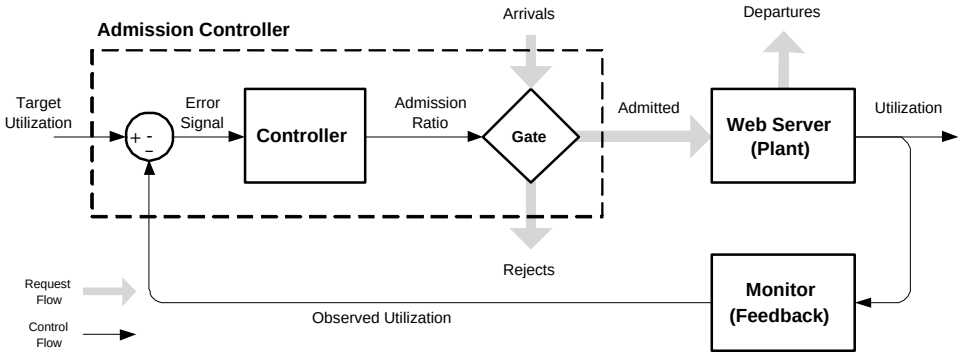

Fig. 3. Block diagram showing the various components of the admission control feedback loop for a web server (as an example of an Internet end-system).

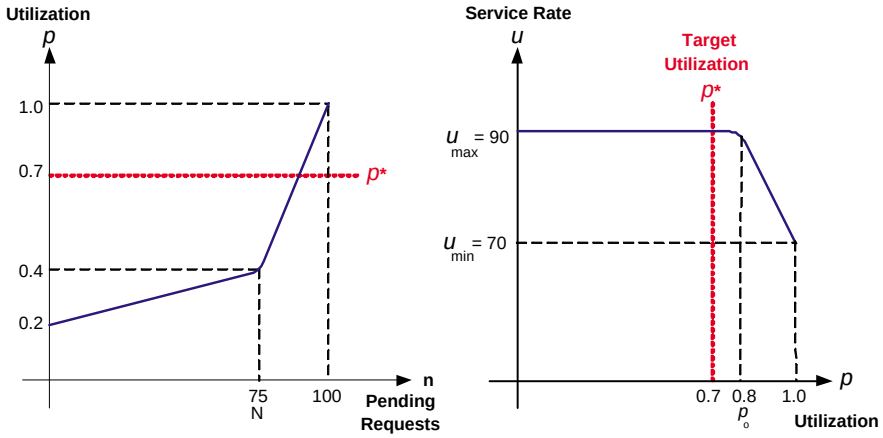

Fig. 4. Instances of (linearized) pricing functions showing the relationships often observed between load and utilization (left) and utilization and thrashing expressed as degradation in service rate (right).

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. Equation 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/sec 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. equation 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 one until the load is low enough to decrease ρ to a value lower than unity. Let's 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, $\delta\alpha(0) + \lambda(\alpha(0) + \alpha(1) ... + \alpha(\theta))$. 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:

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

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

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

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]

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

[3]The numerical solution in the absence of this approximation matches very closely the closed-form solution given in equation 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 rejected R_θ (legitimate) requests. The value of R_θ can be easily derived using equation 9:

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

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:

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

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 equation 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:

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

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:

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

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

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

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 $\bar{\alpha}(0)$, which is close to $\alpha(0)$, starting from $\hat{\alpha}(0)$. Solving for ϕ_2

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

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:

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

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

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

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

$$\pi = \frac{R_\theta + R_{\phi_1} + R_{\phi_2}}{M} \quad (20)$$

B. Numerical Solution

We numerically solve the above system. We assume that legitimate requests arrive at a rate λ of 100 requests per unit time. The number of pending requests $n(.)$ drives $\rho(.)$ as shown in Figure 4(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 Figure 4(right), with $\mu_{\max} = 90$, $\mu_{\min} = 70$, $\rho_o = 0.8$, and $\omega = 70$. Figure 5(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 time units 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 τ=1 second and is repeated every T=50 seconds, producing a potency of over 100, *i.e.*, one request from the attacker results in over 100 legitimate requests being denied service. Figure 5(middle) 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, $\mu_{\min}$. Figure 5(right) show this dependence by showing the attack potency as a function of K for different values of ω. These plots were obtained using equation 20 and the closed-form solutions for R_θ (from equation 12), R_{ϕ_1} (from equation 18) and R_{ϕ_2} (from equation 19). Clearly, the value of K is critical as it reflects the sensitivity of the PI controller to the error signal. Notice that the model we adopted in this section does not capture some other important adaptation parameters. For instance, it assumed 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. The effect of feedback delay will be illustrated in Section IV.

C. Discussion

Measurement-Based On-Line Tuning of RoQ attacks: In the above numerical analysis, we assumed that the attacker knows exactly when the system will recover from the attack so the attacker can correctly time its next burst. In a real setting, the attacker wouldn't know exactly when the system had recovered. A promising approach to correctly time the attack traffic is to use measurements to probe the state of the admission controller. In particular, it is reasonable to assume that the attacker can send few requests and estimate the admission ratio based on its own requests. 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.

Detection and Trace-back: An adversary mounting a RoQ attack does not have to overwhelm the web server 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 a network resource 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.

Tradeoffs Between Efficiency and Tolerance to RoQ Exploits: Tuning an admission controller parameters 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 "normal" operation. For example, as Figure 5(right) suggests, 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. For instance, it may be advantageous for the settings of an adaptation mechanism (*e.g.*, the values of K and $\mu_{\min}$) to be adjusted on-line, based on whether or not the system is suspected to be under a RoQ exploit.

Load Balancing and Profiling as Targets of RoQ Exploits: End-hosts employ other forms of adaptations to ensure scalability [7], [8], [9], [10], [11], [12], [13], [14]. Load balancing is just another form of an adaptation policy that could be exploited using RoQ attacks. There are many approaches to load balancing. However, they all share some commonalities between them (*e.g.*, an inherent feedback delay). Thus, it is possible to cast the load balancing as an optimization process subject to pricing functions (as we have done earlier), which would be vulnerable to RoQ exploits as well.

IV. Internet Experiments

In this section, 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 in this section 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 for the 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

[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.

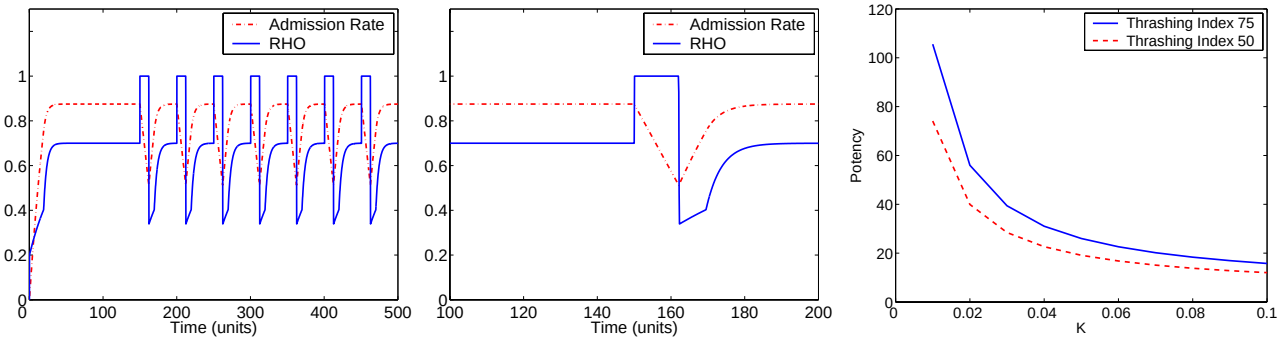

Fig. 5. Assessment of vulnerability to RoQ attacks: Results from the numerical model (left and center) and the effect of K on the Potency (right)

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 doesn't "kill" threads if the system is thrashing. This is important to ensure threads integrity in 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 for 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 show 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 [15], 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 1,024 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 4 machines to generate requests.

Given the above two limitations, the potency values calculated here are lower than those 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 uncommitted transactions to insure database integrity.

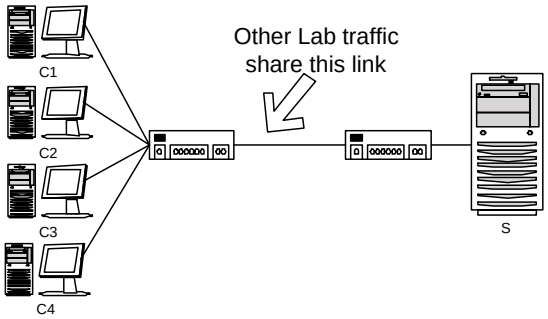

Fig. 6. Experimental setup used in our empirical evaluation.

Experimental Setup: Figure 6 depicts the experimental setup we used in running our experiments. It consists of a server machine (S) running Minihttpd [16] 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 [15] 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 msec to respond to a single request.[6] If the server doesn't 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.

RoQ Exploit of the Admission Controller: Our first set of experiments demonstrates the impact of RoQ attacks on the PI admission controller outlined in Section III. We have instantiated the target of the PI controller to be the memory utilization,

[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.

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.

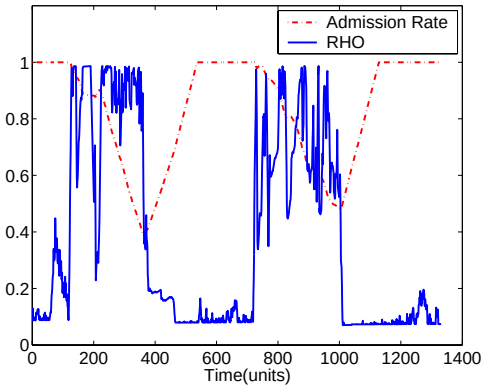

Fig. 7. Admission Ratio and Memory utilization Metric

Figure 7 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 200sec, 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 8 legitimate requests.

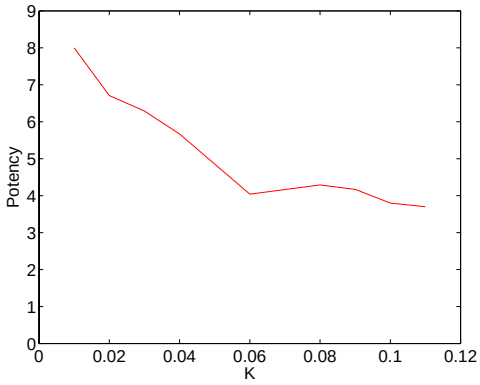

Fig. 8. Potency for different K values

Figure 8 shows the effect of different K values on potency. As K gets larger, the potency decreases. Notice the resemblance in the trend between Figure 7 and Figure 5(left and middle) (obtained analytically), and also between Figure 8 and Figure 5(right) (also obtained analytically).

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. Figure 9 shows how the potency increases as feedback delay increases. When the feedback delay is around 60 seconds, 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.

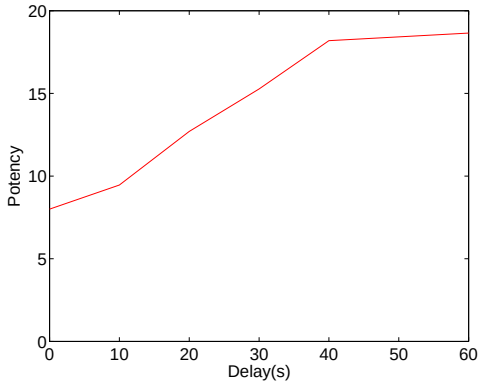

Fig. 9. Potency for different feedback delay

Admission Control versus Concurrency Control: One of the main reasons that makes the admission controller behind Minihttpd [16] susceptible to RoQ exploits, is that it blindly keeps forking threads as requests keep arriving and the system is left at the mercy of its admission controller. Some other web servers, such as Apache [17], maintain a *fixed* thread pool to prevent operating in an overload or a thrashing regime. A Request is only processed when a thread is available. Effectively, this approach limits the level of concurrency (or multiprogramming level) in the system.

Having a fixed thread pool, indeed reduces the impact of RoQ attacks, but this comes at a cost—there is a serious risk that the web-server might become underutilized. Since requests could have very different characteristics, in terms of 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 current requests' profiles [18], [19].

Having a dynamically-adjusted thread pool size is simply another paradigm for implementing admission control. Indeed, thread pool size adaptation could be exploited through RoQ attacks by forcing the thread pool size to oscillate between its two extremes (for example, by alternately subjecting the system to short requests, that don't 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?

Admission Control versus Buffering: In our analysis and experiments, we have assumed that upon the onset of thrashing, requests are "rejected". An alternative approach would have been to simply delay (or buffer) the admission of such requests to the system. Buffering of requests is akin to smoothing the burstiness of the workload. Thus, assuming that the long-term average of the offered load (including that of the attack traffic) is below the capacity of the system to efficiently service that workload, buffering (i.e., delayed service as opposed to denial of service) could be an effective defense against RoQ exploits. This is true, but only if the delays that result from buffering could be tolerated. For many applications, such delays may not be acceptable, or may induce unintended consequences (e.g., triggering timeouts at other layers, which will effectively result in a denial of service). Buffering (as a defense mechanism against RoQ exploits) simply changes the nature of damage—from a reduction in the capacity of the system to service requests to a reduction in the fidelity (response time jitter) of the system. Incidentally, using response time jitter as the damage due to a RoQ attack exploit could well be the goal of the adversary in the first place (e.g., for interactive or real-time applications, including gaming, on-line bidding, etc.)

V. Related Work

To our knowledge, the work presented in this paper is the first to expose vulnerabilities in the *dynamics* of adaptation mechanisms employed in an end-system using a control theoretic framework. That said, 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.

Adaptation through Admission Control: 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 [4], [18]. They are employed in real-time and multimedia systems to ensure that the Quality of Service (QoS) of admitted tasks is not compromised when additional tasks are admitted into the system [20], [21], [22], [23]. They are employed in web/media server designs to ensure that a maximum response time is not exceeded [13], [24], [25], [26], [27], [28]. Other examples are abound [29], [30], [31]. 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 admission controller to ensure the quiescent operation of the end-system behind it.

Control-Theoretic Modelling and Analysis: Marshaling techniques from control and optimization theory has been a fruitful direction as evidenced by the works in [32], [33], [34], [35], [36], [19]. In that respect, we single out the works in [32], [33], 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 [34], 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 [36], 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 [35], 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.

RoQ Versus other Attacks: DoS attacks [37], [38] and its many variants [2] 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 [39], [40], [41]. Using our model, DoS attacks could be classified as RoQ attacks with an infinite aggressiveness index (defined in Section II), 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 [42] 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 are limited to targeting TCP flows only and those who employ the timeout mechanism. RoQ attacks have much broader scope for targeting adaptation mechanisms that can be found in modern computing systems.

VI. Conclusion

In this paper, we exposed a variant 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. We have shown that RoQ attacks can indeed introduce significant inefficiencies in an end-system, while evading detection by consuming a small portion of the hijacked capacity over long-time scales. This is achieved through

exploiting the transients of the underlying system adaptation mechanism. Using a control-theoretic model for an admission controller employed in a web server setting, we derived closed formulas to assess the impact of a RoQ attack through the "potency" metric. 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.

We believe that with the proper understanding of the dynamics involved, one can choose between different adaptation strategies based on risks and rewards behind each of them.

REFERENCES

[1] M. Guirguis, A. Bestavros, and I. Matta, "Exploiting the Transients of Adaptation for RoQ Attacks on Internet Resources," in *Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP)*, Oct 2004.

[2] CERT Coordination Center, "Trends in Denial of Service Attack Technology - October 2001," http://www.cert.org/archive/pdf/DoS_trends.pdf.

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

[4] M. Welsh, D. E. Culler, and E. A. Brewer, "SEDA: An Architecture for Well-Conditioned, Scalable Internet Services," in *Symposium on Operating Systems Principles*, 2001, pp. 230–243.

[5] A. Bestavros, N. Katagai, and J. Londono, "Admission Control and Scheduling for High Performance World Wide Web Servers," Tech. Rep. BUCS-TR-1997-015, Boston University, Computer Science Department, August 1997.

[6] K. Ogata, "Modern Control Engineering, 4th Ed.," Prentice Hall, 2002.

[7] L. Cherkasova, "FLEX: Load Balancing and Management Strategy for Scalable Web Hosting Service," in *Proceedings of the Fifth International Symposium on Computers and Communications (ISCC'00)*, July 2000, pp. 8–13.

[8] M. Aron, P. Druschel, and W. Zwaenepoel, "Cluster Reserves: A Mechanism for Resource Management in Cluster-based Network Servers," in *Proceedings of Measurement and Modeling of Computer Systems*, 2000, pp. 90–101.

[9] A. Fox, S. Gribble, Y. Chawathe, E. Brewer, and P. Gauthier, "Extensible Cluster-Based Scalable Network Services," in *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16)*, St. Malo, France, October 1997.

[10] S. D. Gribble, M. Welsh, J. R. V. Behren, E. A. Brewer, D. E. Culler, N. Borisov, S. E. Czerwinski, R. Gummadi, J. R. Hill, A. D. Joseph, R. H. Katz, Z. M. Mao, S. Ross, and B. Y. Zhao, "The Ninja architecture for robust Internet-scale systems and services," *Computer Networks*, vol. 35, no. 4, pp. 473–497, 2001.

[11] V. V. Panteleenko and V. W. Freeh, "Instantaneous Offloading of Transient Web Server Load," Proceedings of Sixth International Workshop on Web Caching and Content Distribution (WCW '01), June 2001.

[12] J. Chuang, "Distributed Network Storage Service with Quality-of-Service Guarantees," in *Proceedings of Internet Society INET'99*, San Jose, CA, June 1999.

[13] T. F. Abdelzaher and N. Bhatti, "Web Content Adaptation to Improve Server Overload Behavior," *Computer Networks*, vol. 31, no. 11–16, pp. 1563–1577, 1999.

[14] Y. Lu, T. Abdelzaher, C. Lu, L. Sha, and X. Liu, "Feedback Control with Queueing-Theoretic Prediction for Relative Delay Guarantees in Web Servers," in *Proceedings of Real-Time and Embedded Technology and Applications Symposium*, Toronto, Canada, May 2003.

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

[16] "mini_httpd: small HTTP server," http://www.acme.com/software/mini_httpd.

[17] "Apache HTTP Server," http://httpd.apache.org.

[18] M. Welsh and D. Culler, "Overload management as a Fundamental Service Design Primitive," in *Proceedings of the Tenth ACM SIGOPS European Workshop*, Saint-Emilion, France, September 2002.

[19] Y. Diao, N. Gandhi, S. Parekh, J. Hellerstein, and 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, April 2002.

[20] E. Knightly and N. Shroff, "Admission Control for Statistical QoS: Theory and Practice," *IEEE Network*, vol. 13, no. 2, pp. 20–29, 1999.

[21] S. Chatterjee and J. K. Strosnider, "A Generalized Admissions Control Strategy for Heterogeneous, Distributed Multimedia Systems," *ACM Multimedia*, pp. 345–356, 1995.

[22] T. Chiueh and M. Vernick, "An Empirical Study of Admission Control Strategies in Video Servers," in *Proceedings of the 1998 International Conference on Parallel Processing*, Minneapolis, MN, August 1998, pp. 313–320.

[23] S. Son and 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, June 2002.

[24] X. Jiang and P. Mohapatra, "An Aggressive Admission Control Algorithms for Multimedia Servers," in *Proceedings of International Conference on Multimedia Computing and Systems*, 1997, pp. 620–621.

[25] 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, May 2002.

[26] L. Cherkasova and P. Phaal, "Session Based Admission Control: a Mechanism for Improving the Performance of an Overloaded Web Server," Tech. Rep. HPL-98-119, HP Labs, June 1998.

[27] L. Cherkasova and P. Phaal, "Predictive Admission Control Strategy for Overloaded Commercial Web Server," in *Proceedings of 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, August 2000.

[28] T. Voigt and P. Gunningberg, "Handling Multiple Bottlenecks in Web Servers Using Adaptive Inbound Controls," in *Proceedings of Protocols for High-Speed Networks*, 2002, pp. 50–68.

[29] B. Urgaonkar, P. Shenoy, and T. Roscoe, "Resource Overbooking and Application Profiling in Shared Hosting Platforms," in *Proceedings of the 5th symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, December 2002.

[30] S. Jamin and S. J. Shenker, "Measurement-based Admission Control Algorithms for Controlled-load Service: A Structural Examination," Tech. Rep. CSE-TR-333-97, University of Michigan, April 1997.

[31] Z. Tur, A. Veres, and A. Ol, "A Family of Measurement-based Admission Control Algorithms," in *Proceedings of Performance of Information and Communication Systems*, Lund, Sweden, May 1998.

[32] M. Andersson, M. Kihl, and A. Robertsson, "Modelling and Design of Admission Control Mechanisms for Web Servers using Non-linear Control Theory," in *Proceedings of ITCom*, September 2003.

[33] A. Robertsson, B. Wittenmark, and M. Kihl, "Analysis and Design of Admission Control Systems in Web-server Systems," in *Proceedings of American Control Conference (ACC)*, June 2003.

[34] T. Abdelzaher and C. Lu, "Modeling and Performance Control of Internet Servers," in *Proceedings of the 39th IEEE Conference on Decision and Control (ICDC)*, Sydney, Australia, December 2000.

[35] J. Carlstrom and R. Rom, "Application-aware Admission Control and Scheduling in Web Servers," in *Proceedings of Infocom*, 2002.

[36] S. Lim, C. Lee, C. Ahn, C. Lee, and 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, Florida, April 2002.

[37] CERT Coordination Center, "Denial of Service Attacks," http://www.cert.org/tech_tips/denial_of_service.html.

[38] CERT Coordination 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.

[39] A. Hussain, J. Heidemann, and 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*. 2003, pp. 99–110, ACM Press.

[40] J. Mirkovic, J. Martin, and P. Reiher, "A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms," Tech. Rep. 020018, Computer Science Department, University of California, Los Angeles.

[41] C. Meadows, "A Formal Framework and Evaluation Method for Network Denial of Service," in *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, June 1999.

[42] A. Kuzmanovic and E. Knightly, "Low-Rate TCP-Targeted Denial of Service Attacks (The Shrew vs. the Mice and Elephants)," in *Proceedings ACM SIGCOMM'03*, karlsruhe, Germany, August 2003.