Reduction of Quality (RoQ) Attacks on Dynamic Load Balancers:

Vulnerability Assessment and Design Tradeoffs †

MINA GUIRGUIS
msg@txstate.edu
Computer Science Department
Texas State University
San Marocs, TX 78666, USA

AZER BESTAVROS IBRAHIM MATTA
{best,matta}@cs.bu.edu
Computer Science Department
Boston University
Boston, MA 02215, USA

YUTING ZHANG
yzhang@allegheny.edu
Computer Science Department
Allegheny College
Meadville, PA 16335, USA

***Abstract*—One key adaptation mechanism often deployed in networking and computing systems is dynamic load balancing. The goal from employing dynamic load balancers is to ensure that the offered load would be judiciously distributed across resources to optimize the overall performance. To that end, this paper discovers and studies new instances of Reduction of Quality (RoQ) attacks that target the dynamic operation of load balancers. Our exposition is focused on a number of load balancing policies that are either employed in current commercial products or have been proposed in literature for future deployment. Through queuing theory analysis, numerical solutions, simulations and Internet experiments, we are able to assess the impact of RoQ attacks through the potency metric. We identify the key factors, such as feedback delay and averaging parameters, that expose the trade-offs between resilience and susceptibility to RoQ attacks. These factors could be used to harden load balancers against RoQ attacks. To the best of our knowledge, this work is the first to study adversarial exploits on the dynamic operation of load balancers.**

***Index Terms*—Security; Denial of Service; Load Balancing; Performance Evaluation.**

I. INTRODUCTION AND MOTIVATION

To deal with the open-access nature of the Internet, Web applications and services typically employ different forms of adaptation mechanisms to maintain a high-fidelity operation. Of the most commonly deployed forms of adaptation mechanisms are admission controllers and load balancers. Such mechanisms have grown to be very sophisticated in their design, without the proper attention been given to their security aspects. Recent work of ours have demonstrated an instance of RoQ attacks on admission controllers [17]. In this paper, we discover and assess another instance of RoQ attacks on dynamic load balancers.

The Premise of RoQ Attacks: RoQ attacks are a relatively new breed of attacks that target adaptation mechanisms with the premise to hinder an adaptive component from converging to steady-state. In an abstract way, resource adaptation can be viewed as the process of measuring the offered load and setting a price, based on a pricing function. Consumer adaptation, on the other hand, would be the process of observing the price and adjusting the demand accordingly. A price is simply a measure of congestion. RoQ attacks are orchestrated in such a manner that would keep disturbing the prices fed-back to the adaptation mechanisms, resulting in a continuous operation in a transient state. The impact of RoQ attacks is assessed through the "Potency" metric, which reflects the trade-offs between the damage caused by an attack to the cost of mounting the attack. For more information on the theoretical grounds of RoQ attacks, we refer the reader to our work in [16], [17].

Adversarial Exploits of Dynamic Load Balancing Mechanisms: 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 [11], [12], routing protocols [15], firewalls and traffic shapers [32], [14], HTTP and database server farms [27], [26], [18], among others.

In general, load balancing mechanisms could be classified into two categories: static and dynamic. *Static load balancers* either use static information (such as the IP address/prefix of a client or a source) or else they use a prescribed assignment strategy (such as round-robin or weighted round-robin) to make load distribution decisions [14], [11]. *Dynamic load balancers*, on the other hand, rely on metrical statistics fed-back from the resources they manage to dynamically adjust assignment decisions. There is a large body of research (as well as empirical evidence) that has established the poor performance of static load balancers, especially for workloads that exhibit significant variability and/or burstiness. Early examples of such studies include [30], [13], [4]. This realization has led to a large body of research works on dynamic load balancing such as [5], [9], [10], [1], as well as products [19], middleware software ad-ons [33], and entire industries [2].

Invariably, a dynamic load balancer will rely on *delayed feedback* from underlying resources—*e.g.*, utilization, response time, or average number of pending requests at each one of the set of servers/links/routes it manages. Due to the bursty nature of arrivals and service times of the various requests in the workload, load balancers cannot rely on metrical statistics computed over very short time scales, but rather must rely on "smoother" aggregates, which are typically computed over longer time scales, or through Exponentially Weighted Moving Averages (EWMA). Moreover, the fact that the entities performing load balancing are typically not collocated with the resources being managed necessitates the existence of some delay between when

† This work was supported by NSF awards CNS Cybertrust Award #0524477, CNS NeTS Award #0520166, CNS ITR Award #0205294, and EIA RI Award #0202067, and by grants from Fortress Technologies.

metrics are computed and when they are available for load balancing decisions.

The feedback delay inherent in the design of any dynamic load balancer constitutes the "Trojan Horse" through which an a RoQ attack would be mounted. Consider a simple setup of two servers and a load balancer. Upon arrival to the load balancer, a request is forwarded to the least loaded server for processing. Periodically, the servers relay their load to the load balancer. To prevent oscillations between full load and no load, the load value is smoothed out using a weighted moving average. To unbalance the load, the attacker would inject a burst of requests in a very short period of time. Due to the delay it will take the load balancer to register the load differential between the two servers, most of this burst will likely be forwarded to one of the servers, pushing it into overload. Eventually, the load differential will register at the load balancer, resulting in all (legitimate) requests to be routed to the other server. Given that one server cannot handle all legitimate requests by itself—otherwise there will be no need to load balancing in the first place—it is likely that the second server will also be pushed into overload. Once the two servers recover from the ill-effect of this exploit, an attacker would simply repeat its attack.

In addition to the feedback delay, most current load balancers employ a persistence feature (sticky connections) to ensure that connections originating from the same client would always "stick" to the same server, which is mandatory in secure and commercial applications [34]. This feature, however, can enable an attacker to bypass the load balancer, by injecting a burst of requests that would go directly to a single server.[1] Of course, if the servers have public IP addresses, the attacker can simply send the burst directly to the intended server.

Throughout this paper, we limit our exposition to load balancers employed in server farms. However, we believe that the methodology we present in this paper can be extended to load balancing solutions deployed in other settings.

Paper Outline: In Section II, we assess the vulnerability of dynamic load balancers against RoQ attacks, under different load balancing policies. We present a dynamic feedback model to inform load balancing decisions. We give an upper and a lower bound on the impact of RoQ attacks. In Section III, we confirm our analysis and simulation via Internet experimental results. In Section IV, we discuss related work and we conclude with a summary in Section V.

II. Analytic Vulnerability Assessment

In this section, we assess the vulnerability of dynamic load balancers against RoQ attacks. We start with a simple, baseline model involving a static load balancer (*i.e.*, without feedback from the servers), which establishes an upper bound on the impact of RoQ attacks. Next, we present a more realistic, dynamic model that employs feedback to inform load balancing decisions. Using that model, we establish a lower bound on the impact of RoQ attacks.

[1]Notice, however, that a legitimate burst from multiple clients would be load balanced across servers, unless least loaded policy is in use.

To assess the vulnerability of a RoQ attack, we follow our definition of *attack potency*, π, proposed in [16], whereby the potency of an attack is the ratio of the *damage* caused by the attack to the attacker's *cost* for mounting the attack.[2]

$$\pi \quad = \quad \frac{\text{Damage}}{\text{Cost}} = \frac{D}{C} \tag{1}$$

A. An Upper Bound on Attack Potency

Consider a simple setup 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.[3] The load balancer picks the server that is to handle an incoming request based on some load balancing policy. By symmetry, assuming that a static policy is used, the load balancer would assign $\frac{1}{N}$ of the arrivals to each server. 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 load whether load balancing is used or not.

Under this static policy, the arrival rate for each server is $\frac{\lambda}{N}$. According to an M/D/1 service discipline, the utilization, ρ, for each server, is given by $\frac{\lambda}{N}T_s$ and the steady state average queue size for each server is simply given by [3]:

$$q \quad = \quad \frac{\rho^2}{2(1-\rho)} + \rho \tag{2}$$

the average response time, using Little's law, is given by:

$$T_q \quad = \quad \frac{q}{\frac{\lambda}{N}} \tag{3}$$

Following the illustrative example we alluded to in Section I, consider a RoQ attack with an *attack burst* of A requests arriving to one of the servers, which is repeated every *attack period* T.[4] Throughout this paper, T is chosen to be large enough to allow the system to converge back to its steady-state behavior before the next attack cycle. As a result of this attack, the queue size of the server under attack will jump instantly, from its average value q to $q + A$.

The victimized server will spend a period of time recovering from the ill effects of this burst, until its average queue size converges back to its normal, steady-state value, q. However, during this recovery time, newly arriving legitimate requests will experience a larger queue size and delay. We approximate this recovery time using a fluid-like model. Thus, the time it takes the server to get rid of this burst and return to its normal operating queue size q is given by:

$$\hat{T} \quad = \quad \frac{A}{\frac{1}{T_s} - \frac{\lambda}{N}} \tag{4}$$

[2]The definitions in [16] allow for an aggressiveness index Ω, which we take to be 1.

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

[4]Due to sticky connections, an attacker can bypass the load balancer.

Requests arriving during $\hat{T}$ will experience, on average, a queue size of:

$$\hat{q} = \frac{A + 2q}{2} \quad (5)$$

and the number of these requests is $\hat{T}\frac{\lambda}{N}$.

These requests will experience a response time, $\hat{T_q}$, which is lower-bounded by $\hat{q} \times T_s$, thus:

$$\hat{T_q} > \hat{q} \times T_s \quad (6)$$

Notice that equation (6) represents a lower bound on the response time for such requests since at least they have to wait, on average, for the average queue before it is possible to start their service. If we ignore the remaining service time for the request being serviced at time of arrival, we arrive to this lower bound. All other legitimate requests simply experience T_q and the number of those requests is $\lambda T - \hat{T}\frac{\lambda}{N}$. Therefor the average response time, over the whole attack period T, can be derived to be:

$$\acute{T_q} = \frac{\hat{T_q} \times \hat{T} + T_q \times (NT - \hat{T})}{NT} \quad (7)$$

We are interested in assessing the damage resulting from this RoQ attack. A natural metric to use is 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:

$$D_A = (\acute{T_q} - T_q) \times T\lambda \quad (8)$$

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:

$$D_S = (\acute{T_q} - \bar{T_q}) \times T\lambda \quad (9)$$

where $\bar{T_q}$ is the average response time these requests would have experienced, if the attack burst was smoothed over time. In particular, $\bar{T_q}$ is given by:

$$\bar{T_q} = \frac{\bar{q}}{\bar{\lambda}} \quad (10)$$

where $\bar{q}$ is given by:

$$\bar{q} = \frac{\bar{\rho}^2}{2(1-\bar{\rho})} + \bar{\rho} \quad (11)$$

where $\bar{\rho}$ is simply $\bar{\lambda}T_s$ and $\bar{\lambda}$ is given by:

$$\bar{\lambda} = \frac{\lambda}{N} + \frac{A}{NT} \quad (12)$$

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.

The cost (to the attacker) can be captured by several metrics, including the attack magnitude, A, or the attack rate, $\frac{A}{T}$. In this paper, however, we 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:

$$C = A \times T_s \quad (13)$$

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).

Notice that our choice of both metrics, D and C, 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 seconds of additional delay get added to the response time for the legitimate requests.

A Numerical Example: Consider a setup with two servers and a load balancer with an average arrival rate of 15 requests per second, following a Poisson process. Each request requires 0.1 seconds of service.[5] Thus, the utilization of each server is given by $\frac{15}{2} \times 0.1$, which is 0.75 and the average queue size is 1.875 as computed from equation (2) and the average response time is 0.25 seconds as computed from equation (3).

Now consider a RoQ attack with parameters, $A = 100$ and $T = 50$, arriving to one of the servers. The time it takes the attacked server to get rid of this burst can be computed from equation (4) and is equal to $\frac{100}{\frac{1}{0.1} - \frac{15}{2}}$, which is 40 seconds. During these 40 seconds, $40 \times \frac{\lambda}{2}$ (*i.e.,* 300) requests arrive. These requests will observe an average queue size of 51.8 (equation (5)) with an average response time of at least 5.18 (equation (6)). In the remaining 10 seconds, the average response time is back to 0.25 seconds. So the average response time, over the whole attack period, is 4.19 seconds.

To compute D_S, we have to compute the average response time when the arrival rate is $\frac{\lambda}{2} + \frac{A}{2T}$, that is the attack traffic is smoothed. For this case, the arrival rate is 8.5 requests per second (equation (12)). Thus $\bar{\rho}$ is 0.85. The average queue size is 3.258 (equation (2)) and the average response time is $\frac{3.258}{8.5} = 0.383$ second (equation (3)). Thus the sensitivity damage, D_S, over the attack period, is $1,379$. The cost of the attack, as defined in equation (13),

[5] Arrival rates are chosen large enough, not be handled by $N-1$ servers and low enough, not to exceed the total service rate of all N servers.

is $A \times T_s = 10$, so the attack potency is 138 computed from equation (1). That is, for every second the attack requests spend getting service, 138 seconds get added to the waiting time for legitimate requests.

If we are assessing the absolute damage, as defined by equation(8), we get a D_A of $1,479$ seconds giving an attack potency of 148—worse than the sensitivity attack potency.

In our analysis, we used a fluid-like model deriving the transient period as in equation (4). In practice, the arrival rate during this period may change due to the stochastic nature of the arrival process. We present here simulation results that relax this assumption. To that end, we ran several simulation experiments with the parameters above for $50,000$ requests. We use their response time to derive damage when the load balancer is under attack, when there is no attack, and when the attack traffic is smoothed out. The attack potency computed with the absolute damage was 158 and with the sensitivity damage was 148. These values are obtained as an average over 10 independent runs.

Notes: There are a few important points that should be pointed out. (1) The above analysis indicates that static balancing (as with a round robin policy) can result in a very high attack potency, making it easy for the attacker to get more mileage (damage) for its traffic. (2) Static balancing (without feedback) tends to give an upper bound for the attack potency. This is so because any other reasonable balancing policy that utilizes feedback should be able to shift some of the subsequent incoming requests to the other under-loaded servers. (3) The difference between D_A and D_S is relatively small. Indicating that the main contributor to the potency is the sensitivity to burstiness as opposed to the presence of the attack traffic.

Attacking more than One Server: The above analysis demonstrates the case of the attacker sending its burst to only one server. The question arises, is this the best attack strategy, given a fixed attack budget of magnitude A? As it turns out, this is the best attack strategy under static load balancing. We omit the proof due to space limitation, but it could be found in [24].

Figure 4(a) 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. Each point is averaged over 10 independent runs. One can see that attacking one server is the best attack strategy under static load balancing. As the results in this paper will demonstrate, the use of feedback by a dynamic load balancer will reduce damage, but will not eliminate it. We establish this point analytically next.

B. Potency for Dynamic Load Balancing

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

$$q_i^n = q_{i-1}^n + \alpha_i^n \lambda - \mu^n \quad (14)$$

where α_i^n 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.[6] Clearly, $\sum_{n=1}^{N} \alpha_i^n$ at any time instant, i, is 1. Equation (14) is bounded from below by 0.[7]

Ideally, servers should measure their load and report that back to the load balancer. In our analysis, we use the queue size (or a function thereof) 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.

The 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.[8] However, due to the feedback delay coupled with the overhead of communicating instantaneous queue sizes to the load balancer, a more smoothed signal should be used. One example would be to compute averages. We let v_i^n denote the average queue size for server n at time i. The average queue size could be computed over a window of time or using EWMA according to the following equation:

$$v_i^n = (1-\gamma)v_{i-1}^n + \gamma q_i^n \quad (15)$$

where γ is the weight given to the instantaneous measure.

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. There are a number of load balancing policies that can be used. We outline here few of the most-commonly used mechanisms [14], [11]:

(1) Proportional-Control Balancing: 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:

$$\alpha_i^n = \frac{1}{N} + \beta \sum_{j=1}^{N} (q_{i-1}^j - q_{i-1}^n) \quad (16)$$

where α_i^n 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 as we will show below. One can easily see that when all servers have the same queue size, α_i^n will be $\frac{1}{N}$. Also, at any time instant i, $\sum_{j=1}^{N} \alpha_i^j$ will be 1. Notice that here we didn't need to use the average value (v_{i-1}^n) since β is taking care of smoothing.

(2) Weighted Balancing: If the load-balancer uses weighted balancing, then α_i^n would be adjusted according to the following equation:[9]

$$\alpha_i^n = \frac{\frac{1}{v_{i-1}^n}}{\sum_{j=1}^{N} \frac{1}{v_{i-1}^j}} \quad (17)$$

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

[7] In our analysis, we assume infinite queue size.

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

[9] We add a small value ϵ to the average queue size for all servers to prevent division by zero. We take ϵ to be equal to 1.

(3) Least-Loaded Balancing: Under this policy, α_i^n would be adjusted according to the following equations:

$$\alpha_i^n = \begin{cases} 1 & v_{i-1}^n < v_{i-1}^j \quad \forall j \in N, j \neq n \\ 0 & otherwise \end{cases} \quad (18)$$

Numerical Solution: We now solve the above difference equations numerically to capture the effect of RoQ attacks on the above load balancing policies.[10] 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.

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).

For simple illustrations, we take N to be 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 seconds, and that the attack burst of magnitude 100 arrives to one of the servers and is repeated every attack period of 50 seconds.

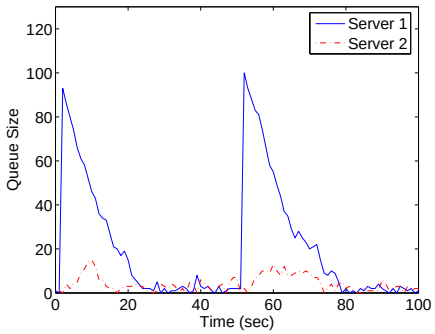


Fig. 1. Simulation results under proportional control

Figure 1 shows the queue sizes for both servers as the attack is launched on server 1, results are obtained using a *proportional-control balancing* policy, with β chosen to be 0.003. Unlike the static balancing policy, 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 seconds (as opposed to 40 seconds, 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.

Similar conclusions are drawn under weighted balancing policy and least-loaded policy, except that the least-loaded policy resulted in higher oscillations due to the on/off nature of the controller. The results are reported in [24].

Figure 2 summarizes the potency values computed as a result of the above attack (simulations are averaged over 10 independent runs). In general, the numerical solution, despite its limitation was able to give a reasonable approximation. Indeed the static case gives an upper bound for any reasonable load balancing.

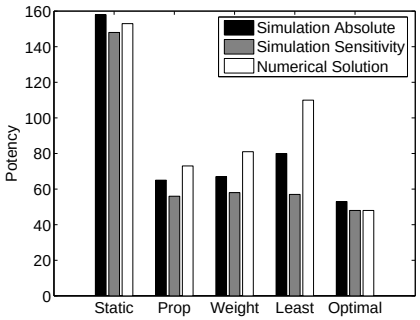


Fig. 2. Summarization of the attack potency for different balancing policies. Simulation results are computed as an average value over 10 independent runs.

Attacking more than One Server: Figure 4(a) shows simulation results obtained from a setup with 6 servers with different values of the parameter β. The attack magnitude A is distributed uniformly at random over the number of attacked servers. One can observe that, with the same attack magnitude, the optimal number of servers to attack is a function of the aggressiveness of the proportional load balancer. When β is small, the performance of the proportional load balancer gets closer to the static policy, indicating that it is better to attack less servers. For example, with $\beta = 0.0003$, attacking two servers is the best attack strategy. If β is large, however, the proportional load balancer can recover quickly and it is better for the attacker to distribute its attack burst over all servers. Notice that in this case, the attacker does not need to bypass the load balancer. As we will show in the rest of this section, there is an optimal load balancer that would achieve a lower bound on the attack potency.

[10]We were able to derive closed-form solutions for the attack potency when proportional control balancing is used, however, we do not present the derivations here due to lack of space, but they can be found in [24].

C. A Lower Bound on Attack Potency

Computing a lower bound on the attack potency is equivalent to finding the optimal load balancing policy (which is simply making instantaneous decisions based on a clairvoyant knowledge of the RoQ attack parameters), even if such a policy cannot be implemented in practice.

Consider our model with N servers. The attack burst arrives to server a while we refer to any other server with h. After the attack burst, server a has a queue size of around $A+q$, thus the optimal load balancer would favor all other servers for the incoming requests.[11] So α_2^a at time instant 2, will jump from its steady state $\frac{1}{N}$ to 0. For all other servers, α_2^h, will jump from $\frac{1}{N}$ to $\frac{1}{N-1}$. The admission controller will remain with such values, until all servers, including the attacked server, reach the same queue size. At this point in time, the optimal load balancer, would return to its steady state giving each server $\frac{1}{N}$ of the arrival rate.

Let τ^* denote the time it takes all servers to reach the same queue size of value q^*+q, given that all incoming requests are being directed to the $N-1$ servers evenly. Thus, for any server h, we have:

$$q^* = (\frac{\lambda}{N-1} - \mu^h)\tau^* \quad (19)$$

The above equation simply states that the accumulation of requests over time τ^* that would result in each server, having q^* additional requests by time τ^*. For the attacked server a, we have:

$$\tau^* = \frac{A-q^*}{\mu^a} \quad (20)$$

The above equation simply states that we have to drain $A-q^*$ requests in τ^* seconds in order to reach a queue size of q^*+q at time τ^*. Equations 19 and 20 could be solved together for the values of τ^* and q^*.

One can capture the damage for legitimate requests, under this optimally-tuned load balancer, by the additional delay that these legitimate requests would experience. Legitimate requests suffer additional delays in two regions. The first region is through the duration τ^* where requests start accumulating at the $N-1$ servers. The total requests arriving during this period is $\tau^* \times \lambda$. The average queue they experience is $\frac{q^*+q}{2}$. Once the load balancer switches the admission ratio back to $\frac{1}{N}$, all servers will spend some time draining their queues from q^* back to q. This time is equal to $\frac{q^*-q}{\mu-\frac{\lambda}{N}}$, which we denote by ϕ. $\lambda \times \phi$ requests arrive during this period and they observe, on average, a queue size of $\frac{q^*+q}{2}$.

We ran simulation experiments to assess the damage under such an optimal (clairvoyant) policy. Figure 3 shows how the queue sizes, for 2 servers, when server a is attacked. The attack is launched with magnitude $A=100$ requests, repeated every $T=50$. The above attack resulted in an attack potency of 53 using the absolute damage metric and 48 using the sensitivity damage metric. These results are averaged over 10 independent runs. The potency we got from the numerical solution was 48. Figure 2 shows the comparison between all policies.

[11] Notice that the ideal load balancing decision is not applied to the attack burst A. As we alluded earlier, in practice, this is possible using sticky connections, for example.

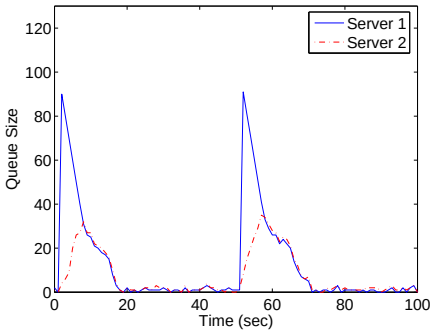


Fig. 3. Simulation result obtained from using the Optimal Balancing Policy with parameters $\lambda=15$, $T_s=0.1$, $A=100$ and $T=50$

Attacking more than One Server: The above analysis can be easily extended to the case the attacker distributes its attack burst over Y servers. In particular, the optimal load balancer would favor the $N-Y$ servers for subsequent legitimate requests until the all servers would have the same queue size. So for any attacked server x, α_2^x will jump from its steady state $\frac{1}{N}$ to 0. For the other $N-Y$ servers, it will jump from $\frac{1}{N}$ to $\frac{1}{N-Y}$. The admission controller will remain with such values, until all servers, including the ones attacked, reach the same queue size. At this point in time, the optimal load balancer, would return to its steady state giving each server $\frac{1}{N}$ of the arrival rate. The best attack strategy under optimal load balancing is to distribute the attack burst over all servers. In that case, τ^* would be 0 and q^* would be $\frac{A}{N}$. Again, the proof can be found in [24].

Figure 4(a) shows simulation results in the case of $N=6$, the optimal curve shows the absolute potency obtained from attacking different number of servers. One can see that attacking all servers, is the best attack strategy, under optimal load balancing.

D. Summary, Practical Considerations and Mitigation

The optimal (clairvoyant) policy we discussed above presents a lower bound on the attack potency whereas the static (inflexible) policy introduced early on in this section presents an upper bound—with all other policies in between.

Notice that the dynamic models proposed do not capture some of the factors that are present in real scenarios such as the presence of feedback delay, the overhead from context switching between requests and the possibility of thrashing as the number of concurrent requests for a given server shoots upwards. We have conducted a series of experiments to quantify the potency of RoQ attacks in the presence of such factors which we report in [24]. We briefly present here some of those results.

Figure 4(b) shows the impact of a 10-sec feedback delay on the absolute potency under the proportional load balancer for different values of β. Notice that large β values that

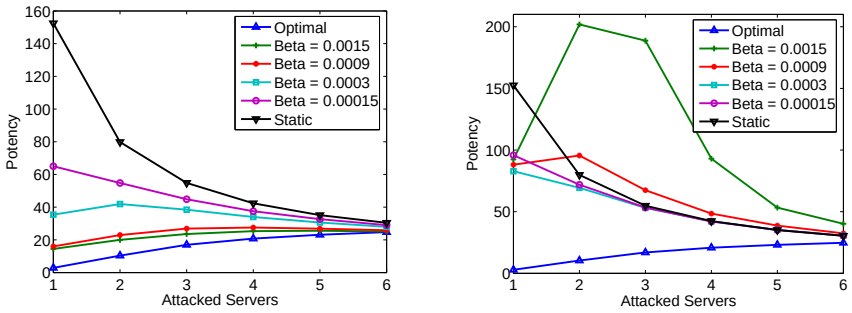


Fig. 4. Vulnerability assessment for proportional load balancer in comparison to the static and the optimal load balancing policies: Without feedback delay (a) and With feedback delay of 10 seconds (b)

decreased the potency close to optimal when no feedback delay was present (Figure4(a)), are now the ones maximizing potency ($\beta = 0.0015$ and $\beta = 0.0009$). That is because the load balancer is reacting in the wrong manner (aggressively), in the wrong time (too late). In these cases, *it is better not to do dynamic load balancing and switch to static load balancing*. If β is small, however, the load balancer reacts slowly and becomes less sensitive to the feedback delay, but at the cost of a potency that is close to the upper bound. Such trade-offs are very important to highlight. Notice also that the damage inflicted is composed of the normal queuing delay due to the additional attack traffic plus the extra queueing delay resulting from the load balancer handling this attack traffic.

In [24], we report on detecting RoQ attacks and the possibility of adjusting some of the parameters online (such as β) to reduce their impact.

III. Internet Experiments

To validate the results we obtained analytically, numerically, and via simulations, we have experimented with a host of web server load balancing mechanisms. Notice that despite the small-scale setup used in our experiments, our results give evidence to the potency of RoQ attacks and they are directly applicable to small-scale services.

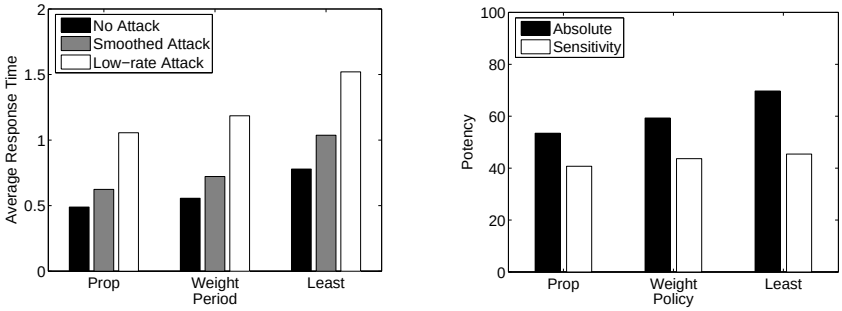
Experimental Setup: Our setup consists of a machine running the load balancer, and two machines running MINIHTTPD[28], and several client machines generating web traffic using HTTPERF[31]. All machines are of 1Ghz/256MB AMD Athlon(tm) and running Linux 2.4.20. We modified MINIHTTPD and HTTPERF to communicate 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 used. As a result, the client initiating the request will establish a normal HTTP connection with the selected MINIHTTPD server. The servers send their feedback information to the load balancer periodically. Linux doesn't 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 to 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 the pure FIFO analytical framework we used in Section II).

In each series of experiments, three scenarios are evaluated: The first measures performance in the absence of RoQ attacks; the second measures performance under a periodic RoQ attack of magnitude A repeated every T seconds; whereas the third measures performance when the attack workload is smoothed out. Each run lasts for 310 seconds. For simplicity, the attack requests are sent to one of the server bypassing the load balancer.[12] In our experiments, 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 [4]. The parameters of the Pareto service time distribution was set to (2,0.05) with an upper bound of 250 seconds and a mean of 100msec. Request arrivals follow a Poisson process with a mean rate of 15 requests/sec. The attack workload was chosen to consist of 100 attack requests that are injected every 50 seconds for five times. All experiments are allowed to warm-up for 60 seconds before measurements were collected.

Susceptibility of Various Load Balancing Policies to Exploits: Figure 5 shows the average response time for legitimate requests under the different load policies under consideration, as well as the corresponding absolute and sensitivity potencies. These results show that the performance of proportional and weighted load balancing policies are similar, and significantly better than that of the least-loaded policy. In our experiments, the proportional policy was slightly better than the weighted policy, especially when the load balancer was under attack. This is due to the feedback delay inherent in averaging the queue size (used in the weighted policy).

Effect of Adaptive Feedback Update Period: In the previous experiments, the MINIHTTPD servers reported back their feedback information to the load balancer every 1 second. This timescale may pose significant overhead, necessitating

[12] As we noted earlier, attack requests can also go through the load balancer using sticky HTTP sessions.



(a) Average Response Time (b) Potency

Fig. 5. Comparative performance and susceptibility to exploitation of the various load balancing policies examined.

the use of longer feedback update periods. The disadvantage of longer update periods is the possibility of a less agile load balancer (under normal legitimate workloads) as well as an increased susceptibility to adversarial exploits. A possible approach to reduce the overhead of feedback signaling without compromising performance or susceptibility to adversarial exploits, is for servers to send the feedback signal to the load balancer whenever the measured signal changes significantly (not periodically). Under proportional control, for example, the instantaneous queue size information is sent over whenever its current value differs from previously the communicated value by more than a given threshold. This threshold can be fixed, or else be a function of the observed average queue sizes in steady state.

To evaluate the effects of the feedback update period as well as this thresholding approach, we performed experiments using fixed feedback delays of 1 second, 3 seconds, as well as a variable feedback update period using a threshold of 5 request differential. Figures 6(a) and 6(b) show the average response times and potencies under these different scenarios, respectively. Figure 6(c) shows the total number of feedback back messages sent by the MINIHTTPD servers to the load balancer. From theses results, we observe that using a fixed feedback period of 3 seconds results in the worst response time and highest potency, but with only $1/3$ of the number of feedback messages sent to the load balancer (compared to the case with a fixed 1-second period). The adaptive feedback period scheme achieves the best balance–being competitive in terms of response time and potencies with the 1-second fixed feedback period approach, and competitive in terms of the number of feedback messages with the 3-second fixed feedback period approach.

Effects of Server Overheads and Thrashing: In previous experiments, the MINIHTTPD servers consumed only minor CPU resources due to sending small files. Thus, the overhead is mainly due to context switching, which is not very large in general, especially when the number of concurrent threads is limited by the thread pool. However, when requests need to consume a large amount of memory or perform significant (disk) I/O, extra overhead due to paging activities become larger and start playing a role in the system performance (and in susceptibility to adversarial exploits).

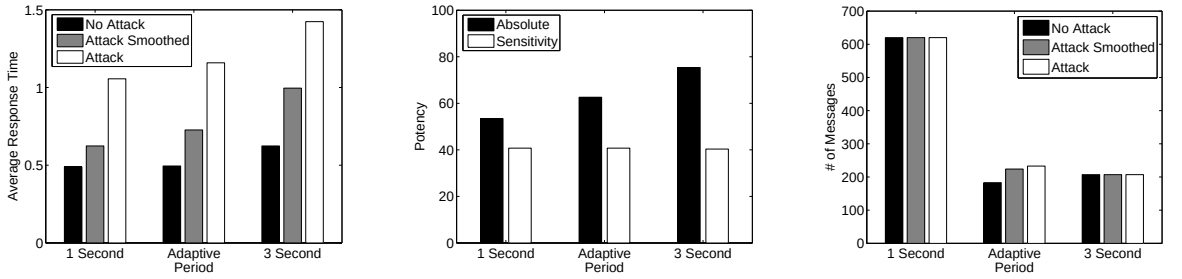
To evaluate such conditions, we required that each request would result in the execution of a CGI program, which randomly reads and writes 6MB chunks of memory. We use the same amount of attack traffic as before on a cluster of 2.40GHz/1.2GB Intel Pentium(R), and we lock 640MB of memory–leaving a total of 560MB for use by the system and for the CGI programs. In these experiments, the average response time for legitimate requests were clocked at 2,412msec when the system is under attack compared to 486msec when the system is not under attack (and 1,046msec when the attack requests are smoothed out). This yields absolute and sensitivity potencies of 192 and 128 respectively, highlighting the fact that server overheads and thrashing would increase attack potencies dramatically.

IV. RELATED WORK

The work presented in this paper relates to three main areas of research: (1) dynamic load balancing, (2) control and queuing theory, and (3) security. In Section I we alluded to a number of studies and products related to dynamic load balancing. In this section, we focus on the latter two areas.

Queuing and Control Theory: There is a huge body of queuing theory literature that studies different queuing disciplines [3] in addition to other research efforts [23], [21] that advance this area further. In this paper, we developed a quasi-static M/D/1-based discrete-time model for load balancing, where the system is assumed to reach steady-state between successive bursts of RoQ attack. This steady-state analysis allowed us to obtain performance measures such as the average queue size at the time of the attack. This is further complemented with transient analysis in the time period immediately following the attack. We used a continuous-time fluid model to obtain transient performance measures such as the time it takes to re-balance the system.

Denial of Service (DoS) Attacks and other Exploits: DoS attacks [7], [6], [8] pose a serious vulnerability for almost every computing system. There are a number of papers that classify various forms of DoS attacks; examples include [29], [25]. Such attacks deny access for legitimate requests by targeting the system's capacity. In comparison to DoS attacks, RoQ attacks do not necessarily result in denial of access, but of service degradation, with an eye on maximizing the response time for legitimate requests. The work presented in this paper, captures current trends in the way attackers are mounting their attacks in order to evade detection. Recent work in [20], shows how attackers are actually moving away from bandwidth floods to attacks



(a) Average Response Time (b) Potency (c) Number of Feedback Messages

Fig. 6. Effect of feedback update periods

that mimic the web browsing of a larger set of clients in order to escape detection. Other work include the low-rate Shrew attacks [22], where low-rate attack traffic can shutoff TCP traffic through targeting the timeout mechanism. That is, in addition, to previous work of ours on the general class of RoQ attacks [16], [17].

V. Conclusion

In this paper, we have exposed new vulnerabilities associated in the operation of dynamic load balancers against new instances of RoQ attacks. In particular, we have shown that due to the presence of feedback delay, RoQ attacks would be able degrade significantly the performance of a dynamic load balancer to the extent that it could be worse than a static one. Persistent connection features, found on dynamic load balancers, have also contributed to the practicality and to the ease in mounting RoQ attacks. Our work examined a number of load balancing policies (similar to the ones being used in practice and those reported in literature) and we assessed the impact of RoQ attacks based on factors, such as the number of resource managed, the feedback delay and the averaging parameters. We believe that such analysis is very important in designing and deploying load balancers as well as in building defense mechanisms against RoQ attacks. Throughout this paper, we supported our analysis with simulations and Internet experiments.

References

[1] W. Aiello, A. werbuch, B. Maggs, and S. Rao. Approximate Load Balancing on Dynamic and Asynchronous Networks. In *Proceedings of the ACM Symposium on Theory of Computing*, 1993.

[2] Akamai. Performance Based Load Balancing. http://www.akamai.com /en/html/services/gtm_how_it_works.html.

[3] A. Allen. Probability, Statistics and Queueing Theory with Computer Science Applications. Second Edition, Academic Press.

[4] M. Balter, M. Crovella, and C. Murta. On Choosing a Task Assignment Policy for a Distributed Server System. *Journal of Parallel and Distributed Computing*, Sep 1999.

[5] V. Cardellini, M. Colajanni, and P. Yu. Dynamic load balancing on web-server systems. *IEEE Internet Computing*, 1999.

[6] 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.

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

[8] CERT Coordination Center. Trends in Denial of Service Attack Technology. http://www.cert.org/archive/pdf/DoS_trends.pdf.

[9] L. Cherkasova. FLEX: Load Balancing and Management Strategy for Scalable Web Hosting Service. In *Proceedings of ISCC*, Jul 2000.

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

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

[12] Cisco. LocalDirector. http://www.cisco.com/warp/public/cc/pd/cxsr/400/index.shtml.

[13] W. Kish D. Dias, R. Mukherjee, and R. Tewari. A Scalable and Highly Available Web Server. In *Proceedings of IEEE COMPCON*, 1996.

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

[15] B. Fortz and M. Thorup. Internet traffic engineering by optimizing OSPF weights. In *Proceedings of IEEE INFOCOM*, 2000.

[16] M. Guirguis, A. Bestavros, and I. Matta. Exploiting the Transients of Adaptation for RoQ Attacks on Internet Resources. In *Proceedings of the ICNP*, Oct 2004.

[17] M. Guirguis, A. Bestavros, I. Matta, and Y. Zhang. Reduction of Quality (RoQ) Attacks on Internet End Systems. In *Proceedings of INFOCOM*, Mar 2005.

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

[19] IBM. High availability load balancing with IBM WebSphere Edge Server for Lotus Workplace. http://www-128.ibm.com/developerworks/lotus/library/edgehighavail/.

[20] S. Kandula, D. Katabi, M. Jacob, and A. Berger. Botz-4-Sale: Surviving Organized DDoS Attacks That Mimic Flash Crowds. In *Proceedings of NSDI*, Boston, MA, May 2005.

[21] L. Kleinrock and R. Muntz. Processor Sharing Queueing Models of Mixed Scheduling Disciplines for Time Shared System. *Journal of the ACM*, 1972.

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

[23] S. Lam and A. Shankar. Response Time Distributions for a Multi-Class Queue with Feedback. In *Proceedings of the International Symposium on Computer Performance Modeling, Measurement and Evaluation*, May 1980.

[24] M. Guirguis. Reduction-of-Quality Attacks on Adaptation Mechanisms. *Ph.D. Thesis. Boston University.*

[25] 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.

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

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

[28] mini_httpd: small HTTP server. http://acme.com/software/mini_httpd.

[29] J. Mirkovic, J. Martin, and P. Reiher. A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms. Technical Report 020018, Computer Science Department, University of California, Los Angeles.

[30] J. Mogul. Network behavior of a busy Web server and its clients. *Research Report 95/5, DEC Western Research Laboratory*, Oct 1995.

[31] 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.

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

[33] O. Othman and D. Schmidt. Optimizing Distributed system Performance via Adaptive Middleware Load Balancing. In *Proceedings of ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM)*, 2000.

[34] Cisco Systems. Sticky Configuration Methods on LocalDirector. http://www.cisco.com/warp/public/117/local_director/sticky_config_ld.pdf.