Model-Based Adaptive DoS Attack Mitigation

Cornel Barna, Mark Shtern, Michael Smit, Vassilios Tzerpos, Marin Litoiu
York University
Toronto, Ontario, Canada
cornel,mark@cse.yorku.ca msmit@yorku.ca bil@cse.yorku.ca mlitoiu@yorku.ca

***Abstract*—Denial of Service (DoS) attacks overwhelm online services, preventing legitimate users from accessing a service, often with impact on revenue or consumer trust. Approaches exist to filter network-level attacks, but application level attacks are harder to detect at the firewall. Filtering at this level can be computationally expensive and difficult to scale, while still producing false positives that block legitimate users.**

This paper presents a model-based adaptive architecture and algorithm for detecting DoS attacks at the web application level and mitigating them. Using a performance model to predict the impact of arriving requests, a decision engine adaptively generates rules for filtering traffic and sending suspicious traffic for further review, which may ultimately result in dropping the request or presenting the end user with a CAPTCHA to verify they are a legitimate user. Experiments performed on a scalable implementation demonstrate effective mitigation of attacks launched using a real-world DoS attack tool.

I. Introduction

DoS attacks have increased in both volume and sophistication [1]. Attack targets include not only businesses and media outlets but also service providers such as DNS, Web portals, etc. A sophisticated DoS attack can be mounted by attackers without advanced technical skills. There are many advanced attacking toolkits freely available on the Internet [2], including LOIC (low-orbit ion cannon, a meaningless nickname) [3]. It has been used to launch attacks on government sites, the RIAA and MPAA (recording and movie industry associations), and more through coordinated efforts such as Operation Payback [3] and Operation MegaUpload. DoS attacks are motivated by a variety of reasons (financial, political, ideological [4]), but regardless of motivation have similar impact: lost revenue, increased expenses, lost customers, reduced consumer trust.

This paper proposes a model-based adaptive architecture and algorithm focused on detecting DoS attacks at the web application level and mitigating them appropriately. A Dynamic Firewall component is added to the standard web application stack; all requests are routed through this firewall. Arriving HTTP[S] requests first encounter a reverse proxy. Here they are filtered based on a set of rules. A decision engine uses a performance model of the application, statistical anomaly detection, and monitoring data from the application to adaptively create, update, and remove rules based on the presence or absence of an attack. As a result of the filtering process, a request is labelled as suspicious or regular. Regular traffic proceeds to the web application as usual, while suspicious traffic is forwarded to an Analyzer component which performs a CAPTCHA test (extended from the work of Morein et al. [5]).

To validate this approach, we implemented the Dynamic Firewall and monitored the traffic to a multi-tier J2EE web application. We monitored the request arrival rate (both suspicious and regular), the response time, and the CPU utilization in regular conditions and attack conditions. The attack conditions were both emulated by a workload generator and created realistically using LOIC. We found that under both attack conditions, our approach adapted to block the DoS traffic, restoring response times to expected values within seconds.

This approach improves on the state-of-the-art in several key dimensions. The adaptive architecture is a novel approach to mitigating DoS attacks. Using statistical anomaly detection to establish a set of filtering rules then iteratively fine-tuning those rules using a two-queuing-network-layer performance model combined with a Kalman filter is a unique application of existing principles. Further, application- and system-level performance metrics, synchronized with the performance model is a novel approach to detecting a DoS attack. "If the traffic exceeds our ability to manage it, there is a DoS attack" is a straightforward definition that allows us to leverage capacity planning work to address this problem. Moreover, filtering traffic using application-level knowledge at the granularity of individual use case scenarios is more granular than typical mitigation approaches. This reduces the impact on legitimate traffic.

The remainder of this paper is organized as follows. Section II reviews relevant work. Section III introduces our adaptive architecture and algorithm. Experiments that showcase the usefulness of this approach are presented in Section IV. Section V concludes the paper.

II. Related Work

We begin with an introduction to DoS attacks and describe some established methods for DoS mitigation. The approach presented in this paper constructs a performance model; while a full review is out of scope for this paper, Section II-B briefly introduces some established achievements in performance modeling.

A. DoS Attacks

A DoS attacker will send many repeated requests that require resources to generate replies. These requests may be low-level TCP requests or higher-level application requests (like GET requests for web pages). The attacker discards the replies, meaning it takes fewer resources to send requests than it does to send responses (this problem is compounded when the target uses SSL; a recently released prototype tool demonstrates a dangerous type of SSL DoS attack [6]). Even with this beneficial ratio, the attacker may not be able to achieve denial of service with a single machine. Distributed Denial of Service attacks harness the power of many distributed attackers to attack a single target; this paper includes DDoS attacks in the term DoS.

The most common DoS attack is a network type of attack. The usual defence is to deploy firewalls and intrusion detection and prevention systems. Firewall rules that implement ingress and egress filtering prevent spoofing attacks that originate on the local network and also prevent incoming traffic from impacting the local network. This impairs the ability of local computers to participate in DoS attacks [2]. Firewalls can also stop TCP-related DoS attacks such as SYN-Floods by implementation of SYN cookies.

DoS attacks which overload computer resources are known to be challenging to defend. Some experts argue the only possible solution for this problem is to improve security for all Internet hosts and prevent attackers from running DoS attacks [2]. An example of these source-end defence schemes is D-WARD [7]. Sachdeva et al. [8] identify a number of problems with source-end defence, principal among them doubt that such mechanisms will be widely implemented.

Researchers who agree with the challenge of defence at the source suggest defence at the victim site. For example, Kargl suggests using available DoS protection tools augmented with load monitoring tools to prevent clients (or attackers) from consuming too much bandwidth [2]. Other examples include QoS regulation [9] and cryptographic approaches [10]. Sachdeva et al. [8] also identified challenges when defending from the victim network, including the computational expense of filtering traffic, the possibility of the defence tools themselves being vulnerable to DDoS, and incorrectly dropping legitimate traffic. While a variety of other approaches have been suggested (e.g., [11], [12], [13], [14], [15]), the current state of the art does not fully mitigate DoS attacks [16].

B. Model

The performance model we use has never been applied to detecting DoS attacks. In our novel application of performance models to DoS attack detection and mitigation, the primary application of the performance model is analysing the incoming traffic and detecting traffic that moves the system toward its saturation point versus traffic that can be handled without overloading the system.

Any hardware-software system can be modeled by two layers of queuing networks [17], [18]: one that describes the software resources and the other one for the hardware resources. This way, the system becomes a network of resources. Each class of service has a *demand* for each resource, which is the time that resource is needed to complete a single user request.

In multiuser, transactional systems, a *bottleneck* is a resource (software or hardware) that has the potential to saturate if enough users access the system. In general, the resource with the highest demand is the bottleneck. However, when there are many classes of requests with different demands at each resource, the situation becomes more complex. A change in the *workload mix* (how users are split among the types of service available) may change the bottleneck, or there may be multiple simultaneous bottlenecks. When a bottleneck saturates, the overall performance of the system degrades quickly, and the system may appear unresponsive.

Early work was done to analyze a system from a performance point of view in [19], [20], [21], [22]. In [23], [24] the authors investigated the influence of workload mixes on the performance of the system, how bottlenecks change with the workload mix and when they become saturated. A method to uncover the worst workload mix and the minimum population required to saturate a system is presented in [25].

Some of the parameters for the model cannot always be measured and other methods are required to find the correct values. Also, the monitored data can contain noise that needs to be removed. In [26], [27], [28] the authors investigated how filters, such as Kalman filters [29], can be effectively used to estimate resource demands. We have used them in our implementation.

III. Adaptive DoS Mitigation

This section introduces our novel approach to mitigating attacks on HTTP web applications[1]. We consider the term *web application* to mean all resources required to run the user-facing components, most commonly HTTP servers, application servers, database servers.

Figure 1 shows our adaptive architecture. To mitigate DoS attacks, we introduce two new components: a Dynamic Firewall and an Analyzer. The Dynamic Firewall is responsible for identifying requests that would potentially overload the server. It implements the adaptive loop: monitoring the performance metrics from the application and the servers, using statistical anomaly detection as a fast heuristic to identify attack trafic and create filtering rules, using a performance model simulation to predict behavior under given traffic

[1]To simplify the presentation, our discussion will refer only to HTTP, but the general approach is also applicable to HTTPS requests.

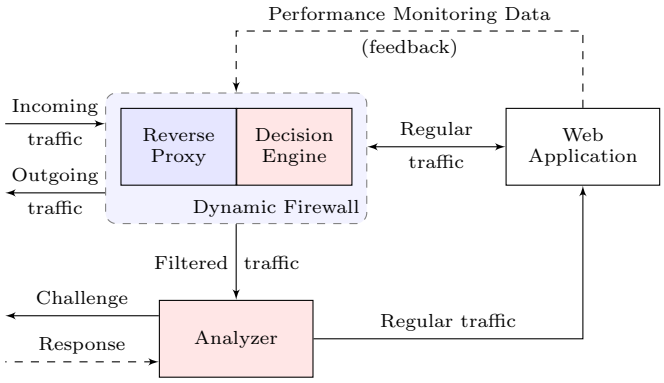


Figure 1: DoS Mitigation architecture.

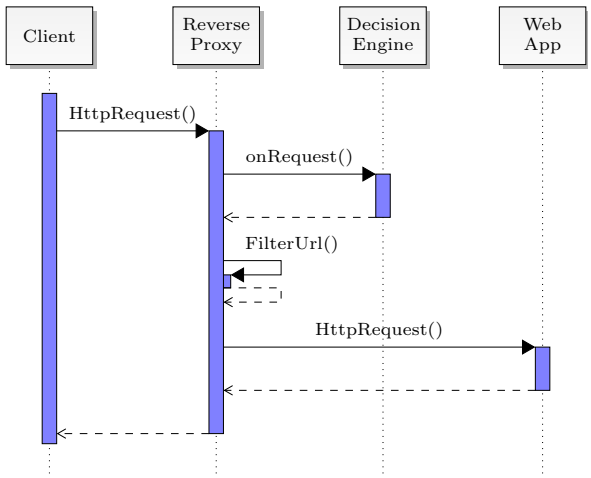


Figure 2: Sequence diagram for handling regular traffic.

conditions and making decisions accordingly, and filtering requests as they arrive based on the adaptively created rules. Incoming requests that are filtered are passed to the Analyzer for further analysis; there, CAPTCHAs are used to identify legitimate requests which are forwarded to the web application, while other requests are assumed to be part of a DoS attack and dropped. Both components are application-aware, and their analysis is sufficiently granular to selectively identify and filter traffic to application components that are being attacked while permitting legitimate traffic to reach the remainder of the web application. Because HTTP is stateless and we do not require session management, the two components can be scaled on-demand.

A. Dynamic Firewall

The Dynamic Firewall (DF) is responsible for redirecting all requests that may overload the server (with respect to CPU, memory or IO). It includes two main components: Reverse Proxy and Decision Engine. The Reverse Proxy is a simple context-aware http/https request router, which redirects legitimate requests to the Web Application and suspicious requests to the Analyzer. Proxy routing is rule-based; the same philosophy as a regular firewall, except that the rules are modified autonomically at runtime by the Decision Engine.

The Decision Engine constructs routing rules for the Reverse Proxy. It constructs a performance model of the web application and then identifies HTTP requests that overload the web application by simulating them on the performance model. In Section III-C, we present the details of the Decision Engine algorithm. Accurate run-time performance simulation requires accurate updates of the current state of the web application (CPU, Memory, IO and application performance counters); the Decision Engine acquires this information.

The high-level operation of the Dynamic Firewall is best described using several use cases.

1) Baseline construction: The Decision Engine detects DoS attacks by utilizing behaviour analysis, specifically a variation of statistical anomaly detection [30]. This analysis requires examples of typical behavior (the baseline). To construct the baseline, it collects OS and Application performance metrics while operating the web application under controlled conditions known to be normal. Using the gathered measurements, the web application performance profile (WAPP) is calculated using statistics-based methods. This automatically constructed profile can be hand-tuned by an administrator to fine-tune the possible ranges of the performance metrics.

2) Regular traffic: The Decision Engine monitors application and OS performance metrics, constantly synchronizing the performance model with the current state of the system. It also analyzes the state of the web application; if it is behaving as usual, the traffic is considered *regular*. Note that a previously-detected DoS attack may still be proceeding, but the rules created when the attack was detected are preventing that traffic from reaching the web application.

Figure 2 shows a sequence diagram for requests arriving while the web application is experiencing regular traffic. When a request is received, the reverse proxy passes it to the Decision Engine which does the filtering step (proceeding similarly to a standard firewall). There the rules relevant to the request are processed and it is forwarded to the web application or to the Analyzer.

3) DoS Attack traffic: When the performance metrics monitored by the Decision Engine deviate from the normals set in the WAPP, the Decision Engine concludes that the web application is under a DoS attack. This triggers the detection of suspicious traffic and the creation of new protection rules for Reverse Proxy. To generate the new protection rules, the traffic collected by the Decision Engine is simulated in the performance model to predict its outcome on the web application. All traffic that causes performance degradation

will be marked as suspicious and corresponding protection rules will be added to the proxy.

Recall that such suspicious requests are not dropped; they are forwarded to the Analyzer for further assessment. The Analyzer will make the final decision on whether to completely drop the request or forward it to the web application. This redirection is not handled internally; rather, an HTTP redirect (302) instructs the client to forward the request to the Analyzer. Typical DoS attack tools are not concerned with the response from the server, and so will not follow redirect requests. This means the majority of DoS attack traffic will be stopped at the Reverse Proxy.

As the DoS attack continues, the Decision Engine is constantly synchronizing the performance model with the web application metrics, and is modifying rules for detecting suspicious traffic. The goal of the adaptation process is to narrow down suspicious traffic detection (reducing false positives). Adaptation also helps respond rapidly to changing tactics by the attacker. The details are presented in Section III-C.

4) Post DoS Attack: The Decision Engine detects that the DoS attack is over when all incoming traffic (at the Reverse Proxy level) could be handled by the web application. The Decision Engine will update the rules of the Reverse Proxy which will stop filtering traffic.

B. Analyzer

The Analyzer is responsible for making the final decision about traffic flagged as suspicious. Our approach is inspired by the process presented in Morein et al. [5]: it presents a CAPTCHA test that must be passed before the request is identified as legitimate. The test is required for each request, to prevent the attacker from passing the CAPTCHA test and then triggering an automatic attack. A pre-filter drops all requests believed to have malicious intent; requests are considered malicious when a request from the same source has failed or not answered the CAPTCHA within a predefined time period. The sequence diagram for the Analyzer is shown in Figure 3.

C. Decision Engine

DoS attacks often succeed when they overload bottlenecked resources, so the decision engine must detect traffic that has the potential to saturate the system. We are using a mathematic performance model to predict how the input traffic will affect system performance. The decision engine implements adaptive loops; Figure 4 provides a conceptual overview of the adaptive loop in the decision engine and its place in the overall adaptive system. The *performance goals* are target performance metrics, such as utilization values, response time, or throughput for a class of request, etc. Those values are derived from the WAPP.

The *decision controller* constructs the filters used by the Reverse Proxy. At each iteration it predicts whether incoming requests would overload the protected web application.

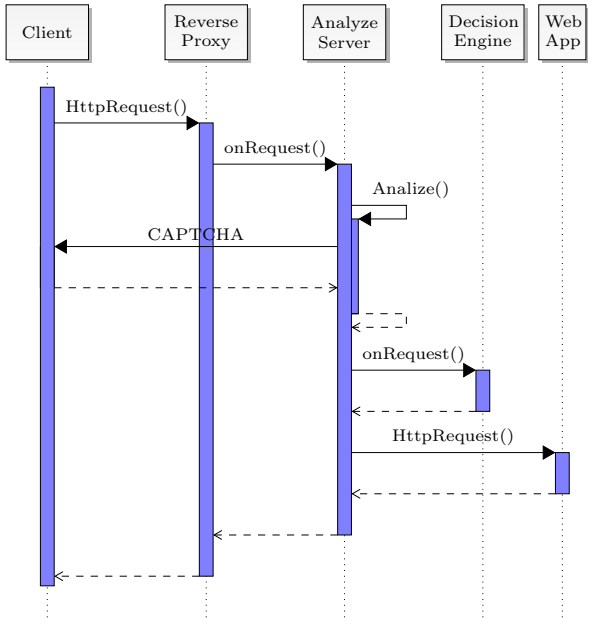


Figure 3: Sequence diagram for traffic redirected to the Analyzer.

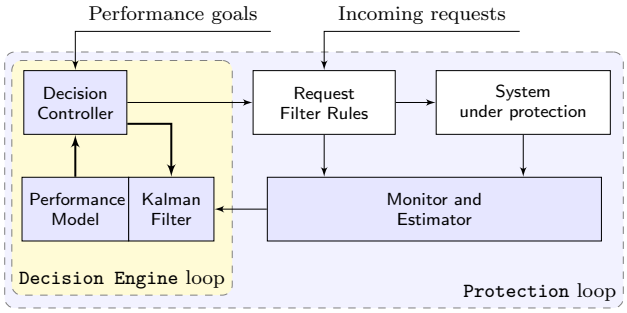


Figure 4: Conceptual Adaptive Model Based Protection

Based on the current state of the web application and the *performance goals*, a new class of requests may be filtered or allowed (filtered traffic is redirected to the Analyzer). Each iteration, the controller tries to minimize the number of classes of filtered requests. The controller relies on a performance monitor for information about the monitored environment and on a performance model for predictions.

The system is continuously monitored by a *performance monitor*. Data includes *CPU utilization*, *CPU time*, *disk utilization*, *disk time*, *waiting time* (which includes time waiting in critical sections, thread pools, connection pools), and *throughput*. The monitor also collects information on the workload and the system. The collected data is filtered through an estimator for error correction and noise removal. Estimators, like Kalman filters [29], have been proven effective in estimating demand [31].

Algorithm 1: Filter Removal Algorithm – algorithm to unfilter a set of scenarios $\mathbb{C} \in \mathcal{C}$ which can overload system

input : $L^{(u)}$ – the vector that contains the current load on each unblocked scenario;
input : $L^{(b)}$ – the vector that contains the current load on each blocked scenario;
input : PM_m – the vector of measured performance metrics;
input : err – the accepted error for the model estimations;
output: $\mathbb{C}$ – a set of blocked classes of service (scenarios), $\mathbb{C} \in \mathbb{C}^{(b)}$, that are safe to unblock.

1 $\mathbb{C} \leftarrow \emptyset$;
2 **if** $\mathbb{C}^{(b)} = \emptyset$ **then**
3 **return** $\mathbb{C}$;
4 Use OPERA to compute the estimated performance metrics, PM_e, for the load $L^{(u)}$;
5 **if** $\left|1 - \frac{PM_e}{PM_m}\right| > err$ **then**
6 Tune the model for load $L^{(u)}$;
7 Compute PM_e with the updated model;
8 $L \leftarrow L^{(u)}$;
9 **while** $\mathbb{C}^{(b)} \neq \emptyset$ **do**
10 $C^{tmp} \leftarrow$ **null**;
11 $pm_e \leftarrow$ **null**;
12 **foreach** *scenario* $C \in \mathbb{C}^{(b)}$ **do**
13 using OPERA model, compute the performance metrics pm_e^C for load $L \cup \{L_C^{(b)}\}$;
14 **if** *(* $pm_e =$ **null** *)* **or** *(* $pm_e^C < pm_e$ *)* **then**
15 $C^{tmp} \leftarrow C$;
16 $pm_e \leftarrow pm_e^C$;
17 **if** *(* $pm_e \neq$ **null** *)* **and** *(* pm_e *are acceptable)* **then**
18 $\mathbb{C} \leftarrow \mathbb{C} \cup \{C^{tmp}\}$;
19 $\mathbb{C}^{(b)} \leftarrow \mathbb{C}^{(b)} - \{C^{tmp}\}$;
20 $L \leftarrow L \cup \{L_C^{(b)}\}$;
21 $L^{(b)} \leftarrow L^{(b)} - \{L_C^{(b)}\}$;
22 **else**
23 **goto** line 24;
24 **return** $\mathbb{C}$;

The performance data is passed to the *performance model*, which consists of two queuing network layers. The main function of the model is to predict performance indicators if currently filtered traffic were allowed, with use-case scenario level granularity. For modeling we are using OPERA, which is an updated version of the solver APERA [32]. Both tools were developed by one of the authors.

The controller creates a filter for requests to a use case scenario when it detects that the nature or volume of the requests deviates significantly from the WAPP, and that system performance indicators are deviating from predefined thresholds. This allows rapid reaction to prevent system overloading. These filters will be iteratively fine-tuned using the performance model.

The second role of the controller is to fine-tune filters or remove them when an attack appears to be over. The filter removing algorithm is shown in Algorithm 1. The algorithm checks if the model is synchronized with the system (line 5), using the specified workload. We say that the algorithm is synchronized if the estimated performance metrics are close to the measured values (within specified error). If it's not synchronized, the model will be tuned using a Kalman filter (line 6).

The main loop (line 9) tries to remove filters. For each filtered scenario, it estimates the performance of the system if the scenario were unblocked. The scenario with the smallest impact on performance is selected (line 14) as a candidate for unblocking. After all scenarios have been evaluated, the algorithm checks if the estimated performance metrics for unblocking the scenario are within acceptable values (line 17). If the test succeeds, the scenario is added to the list $\mathbb{C}$ of scenarios to be unblocked. The traffic for this scenario is taken into consideration for the next iteration of the main loop. The main loop exits when no scenario is found to be a viable candidate for unblocking (line 23) or when all scenarios are unblocked.

IV. Experiments

To validate our approach we conducted three experiments: first using a workload generator to emulate a DoS attack (sudden, heavy load) and using our complete DoS Mitigation strategy, then showing how effectiveness degrades if you exclude the performance model for the same workload, and finally using a popular DoS attack tool in the wild (LOIC) instead of a emulated DoS attack. To measure performance we tracked CPU utilization, request arrival rate and response time. The evaluation is focused on the key contribution of this work, namely the decision-making on arriving traffic; therefore, while in these experiments traffic is redirected to an Analyzer component, that component is not explicitly included in these experiments.

The expected outcome of the experiments is that the model-based adaptive algorithm effectively detects and mitigates the attacks targeting the web application. This will be shown by noting the impact of the DoS attacks on response time and the rate at which regular response times are restored after the start of a DoS attack.

As a testbed, we developed a prototype J2EE web application called *bookstore*. The *bookstore* application emulates an e-commerce web application with the following usage scenarios:

- `marketing` – browse reviews and articles;
- `product selection` – search the store catalog and compare product features;
- `buy` – add items to the shopping cart;
- `pay` – proceed to checkout the shopping cart;
- `inventory` – inventory management such as buy/return items from/to the supplier; and

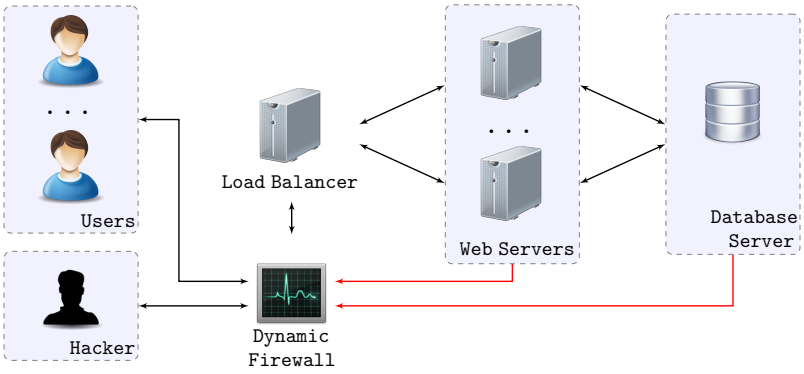
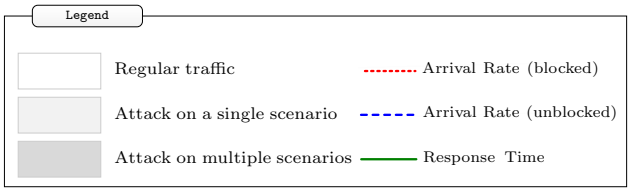
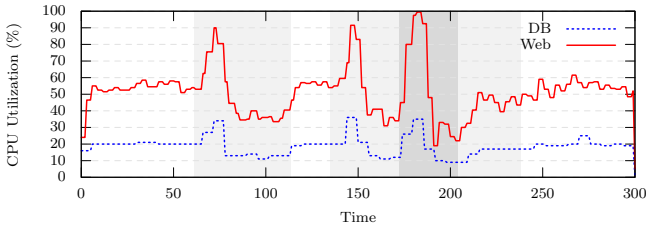
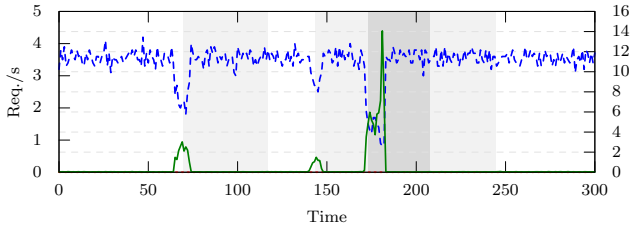


Figure 5: The cluster used for experiments.

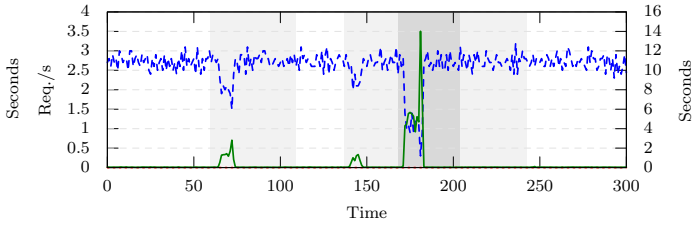




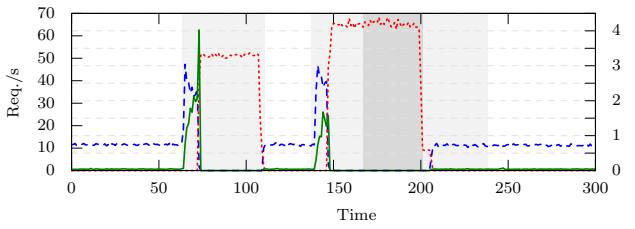
(a) CPU Utilization



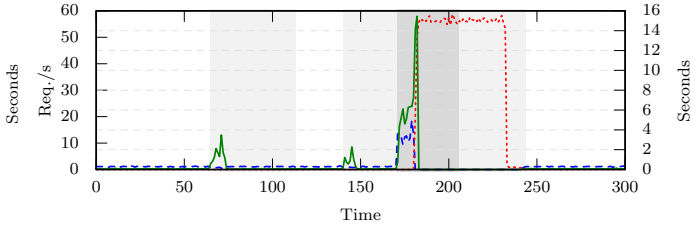
(b) buy



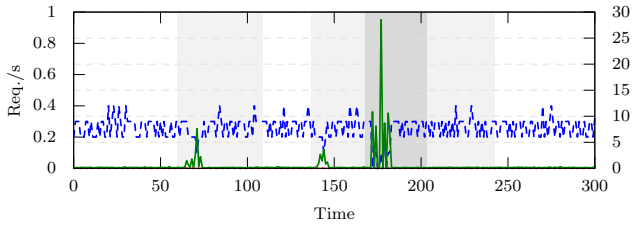
(c) pay



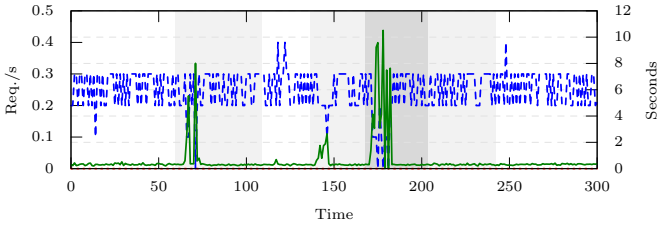
(d) marketing



(e) product selection



(f) inventory



(g) auto bundles

Figure 6: Experiment with emulated DoS attack, using the performance model.

- `auto bundles` – upselling / discounting system.

We deployed *bookstore* to three Windows XP machines: one database server (MySQL) and two application servers (Tomcat). A fourth machine hosted a workload balancer (Apache 2) to distribute the incoming web requests to the application servers. Figure 5 shows our deployment architecture. Once deployed, we gather data under regular conditions to establish the baseline.

For each experiment, there are time periods of regular traffic and time periods with an ongoing DoS attack (noted in the figures with varied backgrounds). We tracked response time (green, solid line), arrival rate of requests that are unfiltered (blue, dashed line), and arrival rate of requests that are redirected to the Analyzer and its CAPTCHA test (red, dotted line).

The results of the first experiment are shown in Figure 6 (with the three metrics for each of the scenarios in Figure 6b-g). There were three attacks: one attacking the marketing scenario, then a second attack on the marketing scenario that overlaps with the third attack on the product selection scenario. The results show that traffic to the attacked scenarios is detected and redirected for further analysis. There is a momentary jump in incoming traffic requests before the attack is detected, then a consistently large number of filtered requests. When the attack ends, the mitigation rules are removed and normal traffic resumes. For all of the scenarios, response time jumps quickly in the seconds before the attack is detected and mitigated. Response time quickly returns to normal once the mitigation action is implemented. Figure 6a shows the CPU utilization on the application servers and the database. There are three spikes which correspond to the DoS attacks. Normal load was restored after the malicious traffic was filtered; based on the performance model, the filter removing algorithm first removed the filter for the marketing scenario and then for product selection.

One of the main advantages of our approach is that while some scenarios are redirected through the Analyzer, the rest are functioning normally, with a response time in the order of tens of milliseconds. Another observation is that the restoration is done smoothly, without oscillations or churn, which indicates we did not remove the filters prematurely.

The second experiment was identical to the first except the decisions were not made using the performance model, and we show only one DoS attack. Figure 7 shows the results, again showing the three metrics for each usage scenario. The algorithm is shown repeatedly filtering and resuming suspicious traffic. Without the performance model, traffic is detected as returning to normal prematurely because the algorithm cannot predict the influence of pass-thru traffic on the overall system performance. Although only one scenario (marketing) is targeted, all of the scenarios experience substantially degraded performance with response times an order of magnitude worse. CPU utilization (Figure 7a) shows that the application servers reached 100% utilization, a sign of being badly overloaded.

Finally, we conducted an experiment using the LOIC tool. As in the first experiment, there were three attacks, the last two overlapping. The results are shown in Figure 8. The mitigations successfully limited the impact of the DoS attack on the web application.

An interesting observation from our results is that under normal conditions, with increasing response time the arrival rate decreases (see Figure 6b) while under attack conditions the arrival rate and response time increase together (see Figure 6d). Higher response time yields more ”think time” from user, because the time in between two requests includes the waiting time for the reply. This is a natural behaviour when the requests are sent by human operators. However an attacker will typically send requests regardless of the response from the web server.

A. Threats to Validity

The experiments conducted used realistic web applications and an actual DoS attack tool. However, with only two experiments using the complete approach there is room for additional validation.

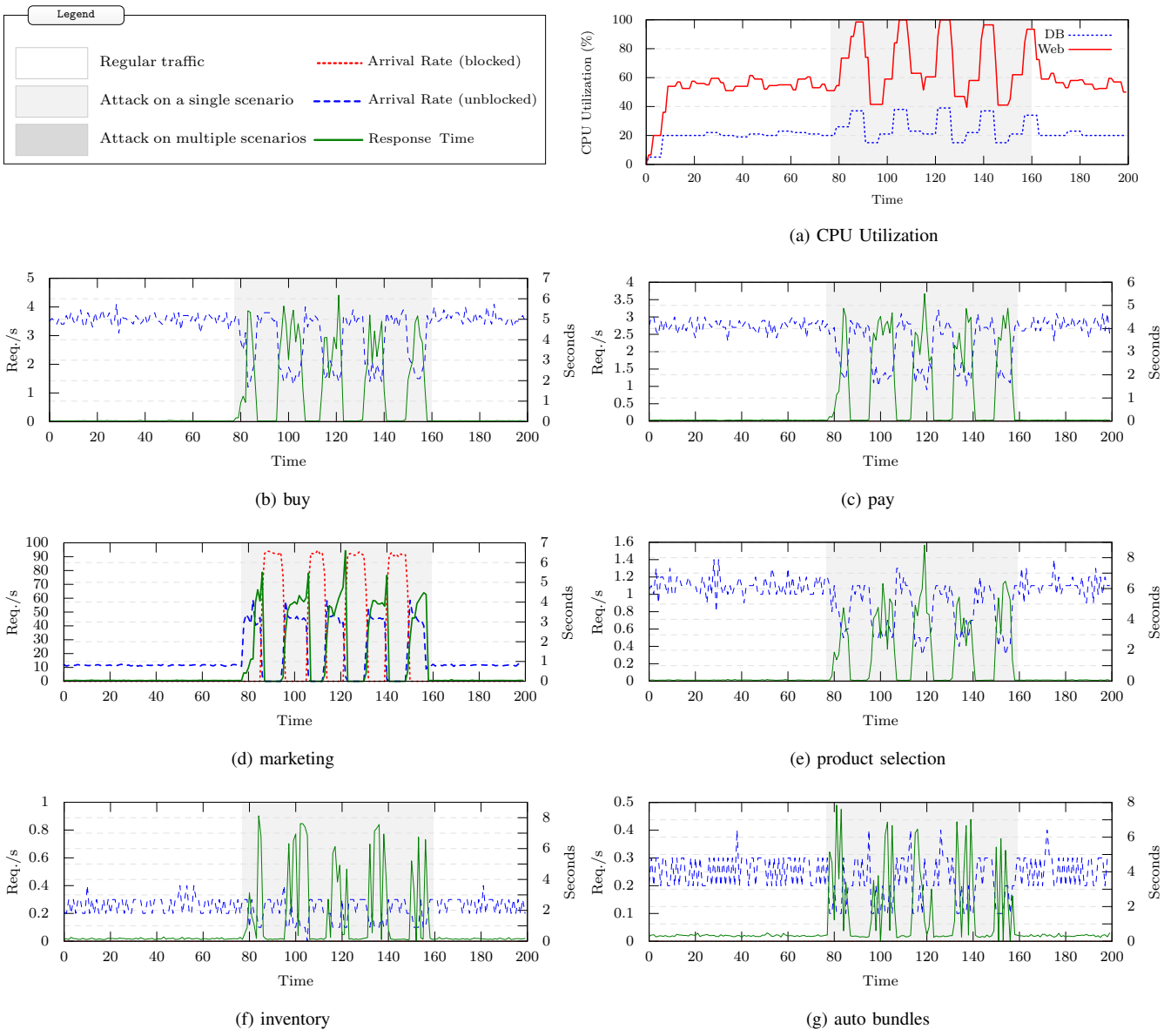
The accuracy of the performance model is an important factor in the accuracy of this mitigation approach.

A false positive is when traffic is detected as an attack incorrectly. Because our approach does not block traffic outright but instead forwards to a CAPTCHA test, that traffic is not lost. However, the test may be annoying to users. There were no false positives in our experiments.

A false negative is when malicious traffic is not detected. In our approach, attacks are only detected when the performance of the application suffers; any malicious traffic that does not have a negative impact will not be detected, but is by definition not a true DoS attack. As our approach blocks types of traffic until the application's performance is acceptable, false negatives will not impact the application.

This approach is intended to address a popular type of DoS attack, a “fast” application-aware attack where the traffic levels increase sharply. This may not be the best approach to mitigate “slow” application-aware attacks where the traffic increases gradually over time; we have not evaluated the performance for this type of attack. As mentioned, we assume existing techniques are in place to defend against attacks not at the application level.

When we compare the effectiveness of our approach to an identical approach that excludes the performance model, we use a threshold model that considers selected performance metrics when making decisions. The selection of metrics used - in our case, CPU utilization, but potentially also including response time, arrival rate, throughput, etc. - will impact the behavior of the system. However, regardless of the metrics chosen, they describe only the current state. Because they are not capable of prediction, there is an



(a) CPU Utilization

(b) buy

(c) pay

(d) marketing

(e) product selection

(f) inventory

(g) auto bundles

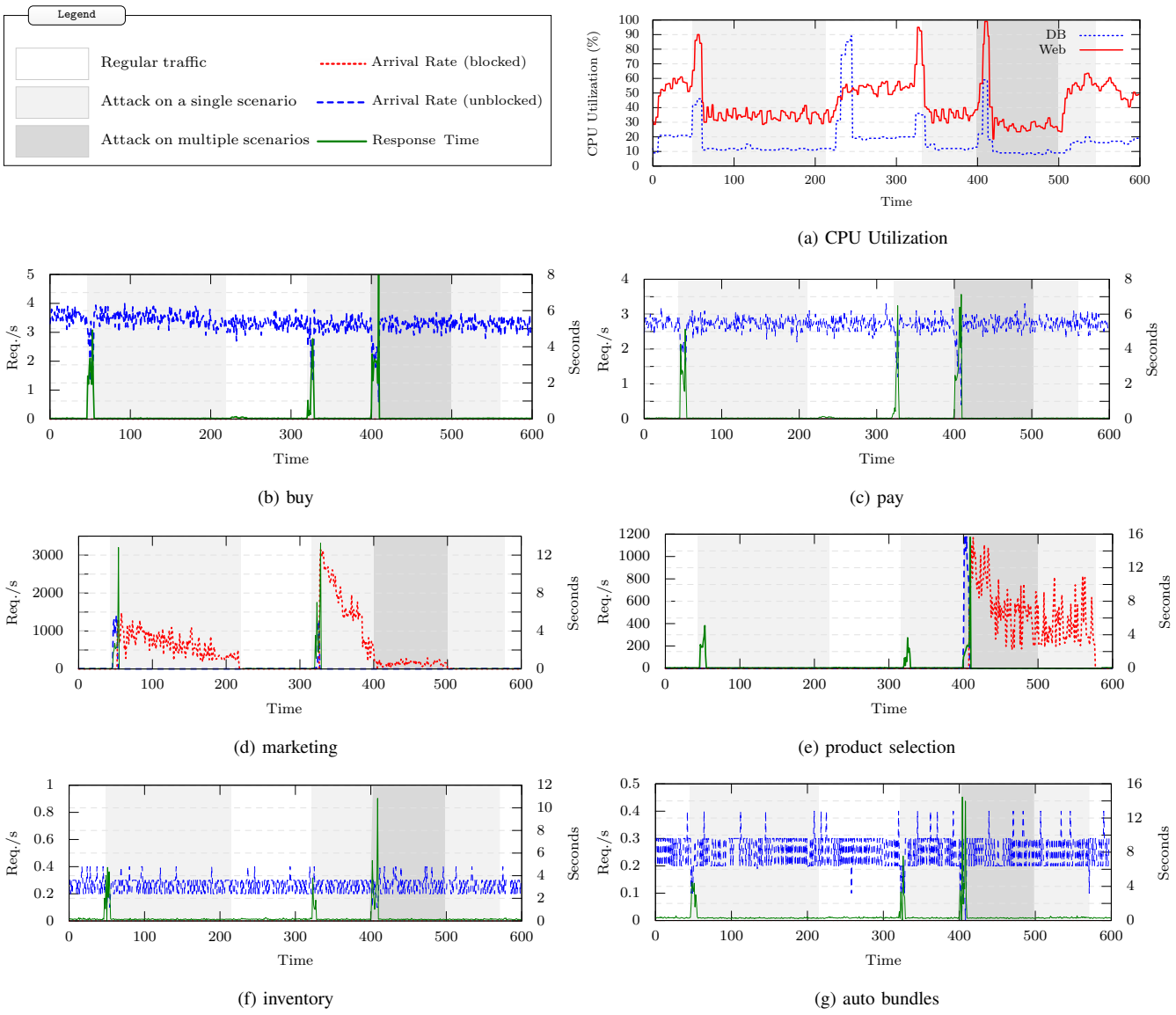
Figure 7: Experiment with emulated DoS attack, without the performance model.

inherent disadvantage to using only threshold models when compared to our combined threshold and performance model approach. Though threshold models may perform as well as our combined approach in certain cases, these models would not work as well across a variety of cases and scenarios.

A DoS attack on one scenario does have an impact on the performance metrics of the other scenarios, before the attack is mitigated. Because the performance of a scenario is a factor in creating filtering rules, this may result in incorrectly filtered traffic to scenarios not under attack. This traffic will have to go to the Analyzer and the legitimate users will need to pass a CAPTCHA test.

V. Conclusion

We have demonstrated an adaptive architecture, an algorithm, and an implementation that effectively detects and mitigates application-aware DoS attacks. The approach, using a performance model and predicting the impact of traffic to create filters to shape that impact until it is at a level the web application can handle, was able to restore normal response times to a web application that was experiencing a DoS attack. It did so efficiently at a use case scenario level of granularity that reduced the impact on legitimate traffic.



(a) CPU Utilization

(b) buy

(c) pay

(d) marketing

(e) product selection

(f) inventory

(g) auto bundles

Figure 8: Experiment with LOIC, using the performance model.

ACKNOWLEDGEMENTS

This work was made possible by funding from NSERC, IBM Centers for Advanced Studies Toronto, and Amazon.

REFERENCES

[1] "Worldwide Infrastructure Security Report," http://www.arbornetworks.com/dmdocuments/ISR2010_EN.pdf, Arbor Networks, Tech. Rep.

[2] F. Kargl and J. Maier, "Protecting web servers from distributed denial of service attacks," 2001.

[3] J. Roman, B. Radek, V. Radek, and S. Libor, "Launching distributed denial of service attacks by network protocol exploitation," in *Proceedings of the 2nd international conference on Applied informatics and computing theory*, ser. AICT'11. Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS), 2011, pp. 210–216.

[4] E. Zuckerman, H. Roberts, R. McGrady, J. York, and J. Palfrey, *Distributed Denial of Service Attacks Against Independent Media and Human Rights Sites*, 2010.

[5] W. G. Morein, A. Stavrou, D. L. Cook, A. D. Keromytis, V. Misra, and D. Rubenstein, "Using graphic turing tests to counter automated DDoS attacks against web servers," in *Proceedings of the 10th ACM conference on Computer and communications security*, ser. CCS '03. New York, NY, USA: ACM, 2003, pp. 8–19.

[6] http://thehackerschoice.wordpress.com/2011/10/24/thc-ssl-dos/.

[7] J. Mirković, "D-WARD: DDoS Network Attack Recognition and Defense," 2002.

[8] M. Sachdeva, G. Singh, and K. Kumar, "Deployment of Distributed Defense against DDoS Attacks in ISP Domain,"

International Journal of Computer Applications, vol. 15, no. 2, pp. 25–31, February 2011, published by Foundation of Computer Science.

[9] A. Garg and A. L. Narasimha Reddy, "Mitigation of DoS attacks through QoS regulation," in *Proc. Tenth IEEE Int Quality of Service Workshop*, 2002, pp. 45–53.

[10] A. Juels and J. G. Brainard, "Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks." in *Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 1999. [Online]. Available: http://dblp.uni-trier.de/db/conf/ndss/ndss1999.html#JuelsB99

[11] T. H. Nguyen, C. T. Doan, V. Q. Nguyen, T. H. T. Nguyen, and M. P. Doan, "Distributed defense of distributed DoS using pushback and communicate mechanism," in *Proc. Int Advanced Technologies for Communications (ATC) Conf*, 2011, pp. 178–182.

[12] S. M. Khattab, C. Sangpachatanaruk, R. Melhem, D. l Mosse, and T. Znati, "Proactive server roaming for mitigating denial-of-service attacks," in *Proc. ITRE2003 Information Technology: Research and Education Int. Conf*, 2003, pp. 286–290.

[13] L. M., W. C.-H. J., J. Y. Hung, and I. J. D., "Mitigating performance degradation of network-based control systems under denial of service attacks," in *Proc. 30th Annual Conf. of IEEE Industrial Electronics Society IECON 2004*, vol. 3, 2004, pp. 2339–2342.

[14] A. K. Pandey and C. Pandu Rangan, "Mitigating denial of service attack using proof of work and Token Bucket Algorithm," in *Proc. IEEE Students' Technology Symp. (TechSym)*, 2011, pp. 43–47.

[15] X. Wu and Y. D. K. Y., "Mitigating denial-of-service attacks in MANET by incentive-based packet filtering: A game-theoretic approach," in *Proc. Third Int. Conf. Security and Privacy in Communications Networks and the Workshops SecureComm 2007*, 2007, pp. 310–319.

[16] P. Jain, J. Jain, and Z. Gupta, "Mitigation of Denial of Servers (DoS) Attack," *IJCEM International Journal of Compuational Engineering & Managment*, vol. 11, Jaunuary 2011.

[17] J. A. Rolia and K. C. Sevcik, "The method of layers," *IEEE Transactions on Software Engineering*, vol. 21, no. 8, pp. 689–700, 1995.

[18] D. A. Menascé, "Simple analytic modeling of software contention," *SIGMETRICS Performance Evaluation Review*, vol. 29, no. 4, pp. 24–30, 2002.

[19] J. Zahorjan, K. C. Sevcik, D. L. Eager, and B. I. Galler, "Balanced job bound analysis of queueing networks," in *SIGMETRICS '81: Proceedings of the 1981 ACM SIGMETRICS conference on Measurement and modeling of computer systems*. New York, NY, USA: ACM, 1981.

[20] D. L. Eager and K. C. Sevcik, "Performance bound hierarchies for queueing networks," *ACM Trans. Comput. Syst.*, vol. 1, no. 2, pp. 99–115, 1983.

[21] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative system performance: computer system analysis using queueing network models*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1984.

[22] M. Reiser and S. S. Lavenberg, "Mean-value analysis of closed multichain queuing networks," *J. ACM*, vol. 27, no. 2, pp. 313–322, 1980.

[23] G. Balbo and G. Serazzi, "Asymptotic analysis of multiclass closed queueing networks: multiple bottlenecks," *Performance Evaluation*, vol. 30, no. 3, pp. 115–152, 1997.

[24] M. Litoiu, J. Rolia, and G. Serazzi, "Designing process replication and activation: A quantitative approach," *IEEE Trans. Softw. Eng.*, vol. 26, no. 12, pp. 1168–1178, 2000.

[25] C. Barna, M. Litoiu, and H. Ghanbari, "Autonomic load-testing framework," in *Proceedings of the* 8^{th} *ACM international conference on Autonomic computing*. ACM, 2011, pp. 91–100.

[26] M. Litoiu, M. Woodside, and T. Zheng, "Hierarchical model-based autonomic control of software systems," *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 4, pp. 1–7, 2005.

[27] M. Woodside, T. Zheng, and M. Litoiu, "The use of optimal filters to track parameters of performance models," in *QEST '05: Proceedings of the Second International Conference on the Quantitative Evaluation of Systems*. Washington, DC, USA: IEEE Computer Society, 2005, p. 74.

[28] T. Zheng, J. Yang, M. Woodside, M. Litoiu, and G. Iszlai, "Tracking time-varying parameters in software systems with extended kalman filters," in *CASCON '05: Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research*. IBM Press, 2005, pp. 334–345.

[29] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[30] S. Oshima, T. Nakashima, and T. Sueyoshi, "Early DoS/DDoS Detection Method using Short-term Statistics," in *Proc. Int Complex, Intelligent and Software Intensive Systems (CISIS) Conf*, 2010, pp. 168–173.

[31] H. Ghanbari, C. Barna, M. Litoiu, M. Woodside, T. Zheng, J. Wong, and G. Iszlai, "Tracking adaptive performance models using dynamic clustering of user classes," in 2^{nd} *ACM International Conference on Performance Engineering (ICPE 2011)*. New York, NY, USA: ACM, 2011.

[32] "Application Performance Evaluation and Resource Allocator (APERA)," 2009, http://www.alphaworks.ibm.com/tech/apera.