Exploiting Kubernetes Autoscaling for Economic Denial of Sustainability

JONATHAN CHAMBERLAIN, Boston University Department of Electrical and Computer Engineering, USA
JILIN ZHENG, Boston University Department of Electrical and Computer Engineering, USA
ZEYING ZHU, University of Maryland Department of Computer Science, USA
ZAOXING LIU, University of Maryland Department of Computer Science, USA
DAVID STAROBINSKI, Boston University Department of Electrical and Computer Engineering, USA

The flexibility and scale of networks achievable by modern cloud computer architectures, particularly Kubernetes (K8s)-based applications, are rivaled only by the resulting complexity required to operate at scale in a responsive manner. This leaves applications vulnerable to *Economic Denial of Sustainability* (EDoS) attacks, designed to force service withdrawal via draining the target of the financial means to support the application. With the public cloud market projected to reach three quarters of a trillion dollars USD by the end of 2025, this is a major consideration. In this paper, we develop a theoretical model to reason about EDoS attacks on K8s. We determine scaling thresholds based on Markov Decision Processes (MDPs), incorporating costs of operating K8s replicas, Service Level Agreement violations, and minimum service charges imposed by billing structures. We build on top of the MDP model a Stackelberg game, determining the circumstances under which an adversary injects traffic. The optimal policy returned by the MDP is generally of hysteresis-type, but not always. Specifically, through numerical evaluations we show examples where charges on an hourly resolution eliminate incentives for scaling down resources. Furthermore, through the use of experiments on a realistic K8s cluster, we show that, depending on the billing model employed and the customer workload characteristics, an EDoS attack can result in a 4× increase in traffic intensity resulting in a 3.6× decrease in efficiency. Interestingly, increasing the intensity of an attack may render it less efficient per unit of attack power. Finally, we demonstrate a proof-of-concept for a countermeasure involving custom scaling metrics where autoscaling decisions are randomized. We demonstrate that per-minute utilization charges are reduced compared to standard scaling, with negligible drops in requests.

CCS Concepts: • **Security and privacy** → **Denial-of-service attacks**; • **Networks** → **Cloud computing**.

Additional Key Words and Phrases: Kubernetes, Economic Denial of Sustainability, Markov Decision Processes, Economics, Stackelberg Security Games

ACM Reference Format:
Jonathan Chamberlain, Jilin Zheng, Zeying Zhu, Zaoxing Liu, and David Starobinski. 2025. Exploiting Kubernetes Autoscaling for Economic Denial of Sustainability. *Proc. ACM Meas. Anal. Comput. Syst.* 9, 2, Article 22 (June 2025), 29 pages. https://doi.org/10.1145/3727114

Authors' addresses: Jonathan Chamberlain, jdchambo@bu.edu, Boston University Department of Electrical and Computer Engineering, 8 St Mary's Street, Boston, Massachusetts, USA, 02215; Jilin Zheng, jilin@bu.edu, Boston University Department of Electrical and Computer Engineering, 8 St Mary's Street, Boston, Massachusetts, USA, 02215; Zeying Zhu, zeyingz@umd.edu, University of Maryland Department of Computer Science, 8125 Paint Branch Drive, College Park, Maryland, USA, 20742; Zaoxing Liu, zaoxing@cs.umd.edu, University of Maryland Department of Computer Science, 8125 Paint Branch Drive, College Park, Maryland, USA, 20742; David Starobinski, staro@bu.edu, Boston University Department of Electrical and Computer Engineering, 8 St Mary's Street, Boston, Massachusetts, USA, 02215.

2476-1249/2025/6-ART22
https://doi.org/10.1145/3727114

1 INTRODUCTION

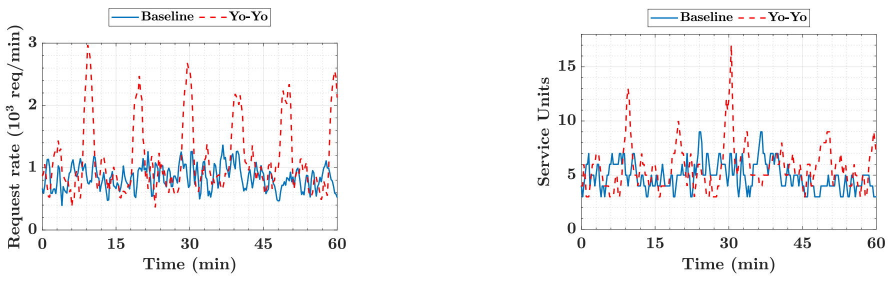
End-user spending in public clouds is projected to reach nearly USD 725 billion in 2025 [23], a level of investment reflective of the entrenchment of cloud technology as foundational to modern infrastructure. This is driven by a number of factors, including Mobile Edge Cloud (MEC) servers offloading computations from networks of Internet of Things sensors [4]; and management of 6G smart city technologies boosting efficiency in urban services ranging from streamlined traffic monitoring, promotion of energy efficiency, and management of urban autonomous devices [4, 33, 57]. Applications operating on cloud-based platforms are highly flexible in their ability to rescale resource ability to match demand. This is particularly evident in commercial settings highly sensitive to temporal demand, *e.g.* video streaming or e-commerce [34], but is observed throughout clouds in general, *e.g.* MEC [4], Wide Area Network fog computing [65], and serverless workloads [25].

This flexibility in allocation also carries a high degree of complexity. While Kubernetes [41] has a built-in Horizontal Pod Autoscaling (HPA) mechanism [40] that is highly configurable in nature, interactions between resources complicates efforts to monitor applications. This has led to the development of tools such as Amazon Web Service's AnomalyMonitor [6] to monitor and optimize cloud financial operations, as well as the emergence of a market for third-party vendors such as CloudZero [16] whose business model is based on outsourcing the problem of optimization. Particularly in the case for applications that run on third-party cloud platforms, this environment results in the potential for exploits. While rapid scaling of additional resources is recommended as a defense against traditional Distributed Denial of Service (DDoS) attacks to prevent service downtime [22], operating the resources to absorb the excess traffic to avoid resource exhaustion incurs additional charges. This can lead to a situation where the adversary's objective becomes the creation of excess resources in order to drive up the target's service bill, potentially to the point of bankruptcy. Such *Economic Denial of Sustainability* (EDoS) attacks can be crafted to pass as a particularly heavy spike in legitimate user traffic [15], increasing the difficulty of implementing filters against EDoS traffic.

One particular class of EDoS attack, known as the *Yo-Yo* [12, 18, 72], periodically alternates between periods of attack and idling: it transmits high amounts of traffic for just long enough such that the auto-scaler creates sufficient replicas to serve the level of traffic being generated, and then goes idle for just long enough for the excess replicas to be discarded. As scale-down delay is longer than scale-up delay in standard autoscaling settings, excess replicas will remain in the system even after the excess traffic stops, and the idle periods will last longer than the attack period. Consequently, a successful Yo-Yo can accomplish relative economic damage comparable to that of a DDoS attack transmitting traffic at an equivalent rate, but at lower cost due to the intermittent nature of the traffic [18]. However, the strictly cyclic nature of the attack, combined with the level of the traffic generated, results in readily detectable patterns, as demonstrated in Figure 1 generated utilizing the experimental Kubernetes cluster detailed in Section 5.

While the Yo-Yo has certain advantages in its simplicity, its detectable nature means it is not necessarily the best option. In addition, a dimension to consider is the nature of the resource being targeted. For many cloud applications, self-hosting is rare due to the capital expenditures required to stand up the necessary hardware. Thus, the actions of third-party hosting services impact the state outcomes. In this work, we aim to formally analyze the following:

- How do charges for resources impact incentives to scale resources in response to changes in request levels?
- How do applications balance tradeoffs between the cost of operating additional replicas and potential Service Level Agreement (SLA) violations?



Incoming Requests Compute Resource Generation

Fig. 1. An example of a Yo-Yo attack [18] against the Kubernetes cluster defined in Section 5 of this paper, compared against baseline utilization during an equivalent service period. The plots demonstrate that both in terms of arrival patterns and the creation of compute resources, the cyclic nature of the attack renders it easily detectable and as such, more sophisticated EDoS adversaries will desire other means of attack.

- Given a particular billing structure and cost considerations, is the optimal scaling policy one that obeys hysteresis conditions? That is, for each level of service (except for the lowest and highest), scale-up and scale-down thresholds exist and are reachable from any arbitrary state.
- Given the target application's scaling policy, how should an adversary launch an EDoS attack in order to maximize its impact?

To determine the thresholds for scaling, we employ Markov Decision Processes (MDPs) [56, Ch. 1] to impose a decision and reward structure upon the dynamic space that is the auto-scaling Kubernetes cloud environment. The attacker leverages an MDP of their own to determine the states where they gain benefit from attacking, given Kubernetes scaling behavior. The result is a Stackelberg Game [17, 21] in two stages, with the defender optimizing first under the belief of a standard workload, and the attacker optimizing in response to the defender's policy. Through this, we model optimal thresholds under a variety of conditions, including whether minimum charges are imposed for adding additional resources, varying thresholds for SLA penalties, and varying transition rates of workload arrivals and service. To realistically evaluate our findings, we run experiments on a Kubernetes cluster running replicas of the TeaStore [71] application within a Docker [19] container, wherein we inject adversarial traffic riding on top of waves of legitimate workload traffic. The efficacy of the attack is judged against multiple potential billing structures, including per-minute utilization and maximum per-hour resource usage charges. Additionally, we leverage the HPA's ability to scale based on custom metrics to introduce scaling based on randomization as a proof-of-concept for potential defense against attacks exploiting the scaling engine itself. Our main results are as follows:

(1) We introduce MDP models, returning corresponding optimal scaling and attack strategies.
(2) Conducting extensive numerical evaluations of the models, we find that optimal scaling policies are generally of hysteresis-type; however, under per-hour minimum charges, scale-down thresholds do not exist, impacting the subsequent attacker strategy.
(3) We demonstrate that if the billing structure does not contain minimum utilization charges, and the cluster contains sufficient request buffer and Service Unit (SU) resources, then independent of other factors the attacker lacks incentive to initiate attacks unless the active SUs equals the maximum available and the request buffer is nearly full.

(4) Conversely, in a resource constrained environment with minimum utilization charges applied, there exist scenarios where it is beneficial for the attacker to launch attacks regardless of the current system state, even if the cluster uses fewer but more powerful SUs compared to a setup with more SUs available.
(5) We demonstrate through experiments on a Kubernetes cluster that charges based on mean per-hour utilization rates limit the impact of attacks on scaling to the greatest extent.
(6) We find that in terms of relative economic damage, there is a limit to which increasing the rate of attacker traffic will increase the attacker reward. In extreme cases, the efficacy of the attack will in fact decrease proportionally to the rate of traffic increase.
(7) As a tool to aid in defending against EDoS attacks, we propose a countermeasure in the form of a custom metric that randomizes the chances of scaling occurring. We demonstrate that accrued per-minute charges under baseline utilization are 90.4% of those under standard HPA, while facing negligible dropped requests.

The remainder of this paper is organized as follows. In Section 2 we review the background and related work on the key areas relevant to this work: Kubernetes and its autoscaling process, EDoS attacks, and MDPs. In Section 3 we define the system and threat models for the cluster model and EDoS adversary. In Section 4 we provide the formal definition of the security game and the associated MDPs, as well as provide numerical evaluation of examples over a variety of configurations to evaluate the impact of components such as minimum utilization charges, the rate of attack (or *attack power* as defined below), and SLA penalty thresholds. In Section 5 we conduct experiments on a realistic Kubernetes cluster configuration, demonstrating the implications of varying billing structures on the resulting cost and relative reward stemming from the adversarial attacks targeting the scaling mechanism, for varying workload types. In Section 6 we evaluate a potential countermeasure built around randomization of auto-scaling.

2 BACKGROUND AND RELATED WORKS

We organize our background survey by the three main topics this work is concerned with: Kubernetes and autoscaling, Economic Denial of Sustainability, and Markov Decision Process modeling.

2.1 Kubernetes and Autoscaling

Kubernetes (K8s) is the *de facto* standard for cloud application deployment, owed in large part to the lightweight nature of K8s pod architecture and the flexibility to scale and re-define container operations as required based on workload demand [41]. This especially holds in environments where consciousnesses of resource consumption is higher such as serverless [25], microservices [71], and Mobile Edge Computing [4]. To accomplish such reallocation in real time production environments [66], Horizontal Pod Autoscaling (HPA) [40] resizes the cluster to the number of replicas required to satisfy a target metric. Methodologies for refining this process is an active field of inquiry, as the inherent complexity of resource interactions spawns a plethora of scenarios to consider [46, 65, 69]. For instance, in a microservices environment, applications are broken down into smaller components that can scale independently; however, this component creates dependencies and the need to monitor multiple resources instead of a single defined whole [46, 48]. Alternatively, serverless environments provide an *illusion* of resource availability but do not necessarily have the commensurate resources active, creating challenges in task scheduling [25, 60].

Another factor to consider is the tradeoff between the cost of operating cluster resources and the tolerance for SLA violations that can result if a cluster does not scale rapidly enough to accommodate traffic increases. One can generally minimize SLA violations or minimize the bill for the compute resources themselves, but not both at once [66]. The nature of the workload will also influence the optimal path to take when attempting to optimize the scaling decision [37, 49]. System workload trace data has been leveraged as part of Machine Learning model approaches to produce predictive thresholds; however, the variety of possible workloads and fact that servers frequently service heterogeneous applications featuring a convolution of such workloads result in the lack of a universal approach [37, 48, 58, 66]. A more comprehensive survey of scheduling algorithm comparisons for Kubernetes in general is contained in [59].

2.2 Economic Denial of Sustainability

Economic Denial of Sustainability (EDoS) attacks are a subset of DDoS-like attacks with the objective of imparting economic damage through triggering excessive resource scaling by the target, and which are classified by stealthier methodologies such as more gradual ramp ups designed to make attacks resemble large spikes of legitimate traffic [15], or bursts of traffic shorter than the traditional DDoS attack length [72]. As such, volume alone cannot be the factor upon which defenses are designed around, a factor compounded by the fact that as of 2021 the majority of DDoS-classed attacks were of the shorter burst type lasting on the order of minutes [31]. DDoS attacks in general are also more likely to be launched during periods when internet traffic is already heavier than normal: during the 2024 Cyber Week period covering 28 November through 2 December - *i.e.* the weekend following American Thanksgiving, a major shopping period - Cloudflare blocked 15% of traffic as DDoS attacks or potential threats in a period where traffic was elevated 12% over the prior week and 40% over the equivalent prior year period [67]. While not all such attacks are EDoS-type, combined with the previous statistic it does raise concerns regarding EDoS, especially for those services relying on third-party hosting platforms (which is most applications), given that cloud is seen as a way to minimize capital expenditures [15].

Attempts to resolve EDoS via Machine Learning techniques have been demonstrated in previous works [3, 30, 47, 72]; however, the models in these works implicitly assume that legitimate user traffic is more or less steady, *i.e.* without significant scaling. One particular class of EDoS attack, the Yo-Yo mentioned in the Introduction, is of particular interest [12, 18, 72], but the strictly cyclic nature of the burst-idle periods renders the attack readily detectable as shown in Figure 1. In addition, the version as presented in [18] also assumes that legitimate traffic is an implicit non-factor and that scaling effects are the result of the Yo-Yo. This ignores bursts from natural spikes in demand, such as during Cyber Week [67], or natural workload variations on a daily or weekly cycle [4]. Further, a dynamic attacker may not be best off under the strictly alternating Yo-Yo, especially if their goal is to amplify traffic effects. It is for these reasons we consider dynamically-scaling scenarios based on the Markov Decision Process model and experiments on a Kubernetes cluster featuring dynamic scaling with varying traffic workloads.

2.3 Markov Decision Process Modeling

Markov Decision Processes (MDPs) provide a means by which to reason about stochastic processes where multiple decisions are possible at a given state, and the next decision depends on the current state [56, Ch.1]. This is accomplished by embedding a Markov Chain with a reward structure, and conditioning the state transitions on the decision made. References [56, 61] provide in-depth treatments of MDPs and their applications. Within the K8s context, an MDP-based model for scaling optimization has been considered previously in [68]. As with this work, the optimization is based on minimization of costs of operating machines and violations from SLAs. Unlike the prior work, our model is structured around demonstrating how the nature of billing impacts the decision making process. As well, we leverage the MDP structure to evaluate the behavior of adversarial agents in the system, which was not a consideration in [68]. The use of MDPs and stochastic games to model interactions among agents more generally is a recurring topic in prior work, owing to the difficulty inherent in making deterministic forecasts in dynamic systems, particularly within economic settings [1, 52, 64].

Stackelberg Games [21], particularly the Stackelberg Security Game subclass [10, 17, 35, 63], are utilized to model interactions within information security settings as a means of characterizing uncertainties in environments and defining placements of defenses. These can be combined with Markovian structures to define the game actions within stochastic spaces [2, 27, 28]. While these works involve the use of MDPs, what makes this work unique is that our model results in a two-stage game with distinct MDPs rather than agents simultaneously improving a common MDP structure. This reflects that the attacker has the ability to optimize with respect to known system behavior, while the defender is unaware of the attacker's presence prior to an attack being launched, resulting in a degree of information asymmetry between players.

3 SYSTEM MODEL

We consider the scenario of a cloud-based application hosted by a platform charging service fees on a *pay-as-you-go* manner. The application seeks to optimize their utilization of cloud resources and therefore minimize the bill from the cloud provider. However, this is balanced against penalty charges for violations of Service Level Agreements with their own customers. As in [68], penalties are incurred if either of the following occur:

i) Requests are not served in a timely fashion, or
ii) Requests are refused by the application altogether.

The former is the result of a request forced to wait beyond some threshold time W for service. The latter is the result of a request arriving to a full buffer. The subsequent scaling thresholds are the result of the interaction of the costs produced from these scenarios being balanced against the charges of operating additional computing resources, represented as arbitrary Service Units (SUs), given both the posted cost of an SU and the nature of the billing structure as noted above.

The underlying hardware infrastructure is relevant for billing purposes; however, as the focus is on the viewpoint of the application, the specific hardware costs are not of concern so much as how the application is billed for usage. Providers vary in both scope of clientele and the variety of configurations offered. Some services round resource billing to the nearest hour, essentially committing projects to minimum charges for newly spun up resources and imposing dead-weight charges for terminated resources not running for a whole number of hours [50]. Others leverage per-second billing and discard unused minutes and hours [8]. Configurations also vary between cluster level [24], per vCPU core [8], and fixed allocation [50]. Therefore, given this variation, we define the Service Unit to generally represent the level of computing power represented in billing. We leverage mechanisms to model charges which apply on a per-second, per-minute, or per-hour basis with minimum costs to operate SUs once active. To facilitate this and relate the costs to the level of SUs and the target scaling thresholds, we employ Markov Decision Process (MDPs) models.

3.1 Markov Decision Process Definitions

MDPs consist of three principal components [61, Ch.5]:

- The *state space* $\mathcal{S}$, describing the system states s that the process can take on.
- The *decision space* $\mathcal{D}$, consisting of the potential decisions d that may be taken at a given state (this is also known as the action space in some sources).
- An *objective function*, the optimization of which for a given $\mathcal{S}$ and $\mathcal{D}$ yields a corresponding optimal policy π consisting of the decisions $d = \pi(s)$ to take at each state $s\mathcal{S}$.

Table 1 contains parameter definitions relevant to the generalized view of our MDP model of the Kubernetes cluster. Specifically, letting M be the limit of SUs that can be associated to the cluster,

Parameter	Definition
$\mathcal{S}$	Space of possible states s that comprise the MDP.
$\mathcal{D}$	Space of possible decisions d to be made at a given state.
$\mathcal{A}$	Space of decisions a made by the adversary.
C	Cost function, relating cost of making decision d at state s.
$\mathcal{R}$	Reward function, relating reward obtained from making decision a at state s.
π	A policy; the optimal decisions to make at state s.
m	Number of active SUs.
M	Cap on active SUs.
n	Number of requests present in the system, both queued and in service.
N	Buffer size of requests.
λ	Poisson rate of arrivals of legitimate workload requests.
μ	Rate of service per Service Unit.
K	Attack power; defines rate of transmission by attacker: $\lambda_a = K\lambda$.

Table 1. Generalized Markov Decision Process parameter and variable definitions.

and N be the maximum request buffer size, then the state space takes the form:

$$\mathcal{S} = \{(m, n) | 1 \leq m \leq M, 0 \leq n \leq N\}. \tag{1}$$

The corresponding decision space $\mathcal{D}$ for the defender consists of whether and how much to scale by at any step. While HPA allows for scaling by arbitrary individual pods, the related Cluster Autoscaler [39], which applies horizontal scaling to full clusters, only allows adding or removing at most 1 cluster at a time. Therefore, we let $\mathcal{D} = \{-1, 0, 1\}$ in general. However, as we must have at least 1 SU, and we have a limit of M SUs, we have special conditions $\mathcal{D}_{(1,n)} = \{0, 1\}$ and $\mathcal{D}_{(M,n)} = \{-1, 0\}$ to remove invalid actions. For the attacker, their decision space is simply $\mathcal{A} = \{0, 1\}$ corresponding to remaining idle, or launching an attack, respectively.

Consistent with other MDP models of Kubernetes clusters [68], we assume that incoming requests arrive according to a Poisson distributed random process with rate $\lambda > 0$. If $a = 1$, the attacker is said to be transmitting an attack with *(attack) power* K, defined as follows:

Definition 3.1. Under *attack power* K, the attacker traffic is a Poisson distributed random process with rate traffic $\lambda_a > 0$, such that $\lambda_a = K\lambda$, *i.e.* the attacker traffic rate is K times higher than the baseline customer traffic's rate λ.

Further, each SU is identical to all others, processing requests according to an exponential distribution with rate $\mu > 0$. These rates will in turn define the probabilities of transitioning between states as outlined in Section 4 and Appendices A-B.

Given the respective objective functions of $C(s, d)$ for the defender representing the cluster operating cost and $\mathcal{R}(s, a)$ for the attacker representing the net reward of launching an attack, and the state transition probabilities, the goal is to determine the set of decisions that optimizes the respective functions. We accomplish this via iteratively solving *Bellman optimality equations*. We specifically employ Discounted Value Iteration [56, Ch. 6] as this has the computational advantage of solving the Bellman Equations in a single step. The *discount* refers to the extent to which solutions are weighted to short-term vs. long-term gains; the full details of the algorithm are given in Appendix C. However, we note that as part of this, as we deal with continuous arrival and departure processes, we must map the MDP onto a discrete solution space, transforming it into a Semi Markov Decision Process (SMDP) [56, Ch. 11].

As we model a Kubernetes cluster, it is especially desirous to establish that a given policy π satisfies the hysteresis hypothesis in the sense of [45], where the thresholds for scaling up and down exist at each level and are reachable from any arbitrary state. This is formally defined below:

Definition 3.2. Given a policy π for a set of states $\mathcal{S}$ and decisions $\mathcal{D}$, define a double-threshold sequence F_m and R_m for scale-up and scale-down as follows for $1 \leq m \leq M+1$ [38]: $1 = R_1 \leq R_2 \leq \cdots R_M \leq R_{M+1} = \infty$, $1 = F_1 \leq F_2 \leq \cdots F_M \leq F_{M+1} = \infty$; and for $2 \leq m \leq M$, $R_m = \min\{n : \pi(m,n) \geq 0\}$, $F_m = \min\{n : \pi(m,n) \geq 1\}$. If the set is empty, the minimum is infinity.

π is a *hysteresis policy* if such a double-threshold sequence exists and the following holds: $R_m \leq F_m$, $R_m \leq F_{m+1}$ for $1 \leq m \leq M$, and for all states s:

$$\pi(s) = \begin{cases} -1 & \text{if } n < R_m \text{ and } m > 1, \\ 0 & \text{if } R_m \leq n < F_{m+1} \text{ and } m > 1, \\ 1 & \text{if } n \geq F_{m+1} \text{ and } m < n. \end{cases}$$

Definition 3.2 provides criteria for assessing whether the defender's given π for the defender is a reasonable policy. However, it does not provide criteria assuring such conditions are fulfilled *a priori*. Thus, it is not guaranteed that the π returned is hysteresis: a policy which is optimal yet violates the hysteresis hypothesis is likely to be the result of maladaptive incentives in the cost structure guiding scaling decisions. From the attacker perspective, such policies can in turn increase the incentive to attack and drive up costs at lower thresholds.

3.2 Threat Model

In defining the threat model, the attacker is assumed to have a degree of system knowledge. The most straightforward means to accomplish this is to conduct probing attacks against the target [70] utilizing tools such as ZMap [20] and infer the system state via relative changes in response times; these methods can be enhanced via applying techniques for scaling, applying response time measurements to estimate if trigger points have been reached [73]. Furthermore, under the assumption of an environment with mobile edge or fog computing characteristics, state awareness could be achieved via a compromised IoT device connected to the network, due to the less-secure software such devices frequently utilize [53, 62]. Clouds servicing mobile devices may also be vulnerable to observation via the wireless aspect, as energy detection can be leveraged as a proxy for traffic [44]. Additionally, methods exist for fingerprinting system configuration changes, and an intruder can potentially repurpose tools to detect changes by discovery [13, 14].

Regardless of methodology, the adversary is able to leverage their knowledge of available tools to observe the system state to some degree, and makes their decision to inject traffic based on the determination of the current state and the perceived utility of the cost of the traffic vs. the cost of the resulting extra computing resources this would generate. Under the (S)MDP model, the attacker makes judgments on a moment-to-moment basis, and as a result we seek a measure of an attack's efficacy to judge whether the higher cost of sending additional traffic is justified in terms of generate additional Service Units relative to a lower attack power. We accomplish this by defining efficiency in terms of excess SUs compared to time of attack and the attack power:

Definition 3.3. The *efficiency* of an attack is defined in terms of the relative damage imparted. Letting $frac_{attack}$ be the fraction of time the attacker chooses to attack, SU_{attack} be the number of SUs generated as the result of attacks, and $SU_{baseline}$ be the number of SUs generated under baseline customer traffic, then for attack power K the corresponding efficiency is equal to:

$$Eff = \frac{SU_{attack} - SU_{baseline}}{K \times frac_{attack}}. \tag{2}$$

3.3 Limitations

While (S)MDPs give the ability to reason about where levels of scaling *should* occur, there are limitations to the approach. For instance, while the transition process is discretized, the HPA control loop in reality is an independent process that runs periodically to determine if scaling is warranted, rather than being triggered instantaneously upon an arrival or departure of a job request [40]. Thus, while policies yield optimal thresholds, it does not necessarily reflect actual behavior as the HPA is not a continuously-running process.

Further, while MDPs have been incorporated with Stackelberg Security Games in prior works [2, 27, 28], our setup is that of a best-response where players optimize their actions independently, rather than a stochastic game where players are optimizing in response to each other's movements [64]. While this is not inherently unreasonable, as a) defenders are not necessarily aware of an attack beforehand and b) scaling thresholds are by default deterministic [40], a stochastic game setup with direct response indeed yields different results. This is particularly true if the defender has the means to update their policies on a relatively short-time scale.

Finally, while we do consider the attacker to incur a cost to attack, we consider the per-stage costs and thus optimization is unconstrained. The interest is to determine the conditions under which an advantage exists to launch an attack at any given moment in time. However, this does not take into account the available attacker budget, although as we reason in terms of kilowatt-hours in our numerical examples in Section 4.3, we are optimizing over an arbitrary hour's length period rather than an indefinite state. Regardless, as decisions in the (S)MDP only depend on the current state, properly accounting for the budget requires a temporal component. If this is not done properly, a *State Space Explosion* is possible, resulting in the MDP becoming intractable. Incorporation of techniques from [11] for optimization of MDPs with Age of Information states, which present a similar problem, has been identified as a potential means to address this in future work.

4 STACKELBERG SECURITY GAME

In this section we define the Stackelberg Security Game, the corresponding Markov Decision Processes governing each player decision, and provide a numerical example of the game evaluation.

4.1 Game Definition

The formal game model definition consists of an *attacker* and *defender* who each possess a set of strategies and a utility function that maps the strategies to a reward. The attacker strategies consist of where to launch an attack, while the defender strategies consist of deploying resources to defend against attacks [35, 63]. In the case of the computing cluster being modeled here, the defender is the application owner, whose strategies consist of where to assign the autoscaling thresholds and minimize operating costs. The attacker is an adversary whose strategies consist of which states to inject additional traffic into, and are rewarded by forcing the defender to increase computing resources and corresponding operating costs at a rate greater than the cost of generating traffic.

Given the nature of the environment, we assume a degree of information asymmetry between attacker and defender. Specifically, the defender is optimizing the scaling thresholds based on typical workload expectations. The attacker, per the threat model and knowledge of Kubernetes infrastructure, is able to infer the system state from observations. This results in a two-stage game. The first stage consists of the defender determining the optimal autoscaling policy based on expected workloads, the pod configuration, and SLA costs imposed for delayed and dropped requests. The second stage consists of the attacker optimizing where to inject additional traffic based on knowledge of defender policy and the corresponding cost for the cluster to enter a given state, balanced against the cost of transmitting the additional traffic. Due to the nature of arrivals

Parameter	Definition
Λ	Total transition rate per state, equal to $\lambda + \min(m, n)\mu$.
$\tilde{\Lambda}$	Transition normalization rate, equal to $\lambda + M\mu$.
$\mathcal{S}^{\pi}$	Subset of $\mathcal{S}$; states reachable from $(1, 0)$ under the optimal policy.
Λ_a	Attacker's total transition rate per state, equal to $(1 + \mathbb{1}_{(a=1)})K\lambda + \min(m, n)\mu$.
$\tilde{\Lambda}_a$	Attacker's transition normalization rate, equal to $(K + 1)\lambda + M\mu$.
C_S	Cost of maintaining Service Units.
C_R	Cost of dropped requests.
C_P	Penalty cost for holding requests in the queue for an excessive duration.
C_D	Cost of surplus time for (de)activating a Service Unit.
C_A	Cost of transmitting adversarial traffic.
W	Threshold for SLA delay violation.
β	Discount factor for Value Iteration solution.

Table 2. Parameters and variables utilized in the specific defender and attacker Markov Decision Process model definitions.

and departures as stochastic processes and the attendant reward structure, we model each stage as a (Semi) Markov Decision Process to determine the corresponding optimal policy.

4.2 Markov Decision Process Models

In this section we provide the definitions for the SMDPs corresponding to each stage, based on the model outlined in Section 3.1. Tables 1 and 2 detail the parameter definitions utilized in the SMDPs.

4.2.1 Defender Stage MDP. Given the state space $\mathcal{S}$ from Equation (1) and the decision spaces for each state $\mathcal{D}_s \subset \{-1, 0, 1\}$, defining the SMDP requires a formal definition of the state transition probabilities, and the cost function $C(s, d)$. For the former, as noted transitions between arbitrary states $s \longrightarrow t \in \mathcal{S}$ will be triggered by a single arrival or a single departure, resulting in rates:

$$\begin{cases} \lambda & \text{if } t = (m + d, \min(n + 1, N)), \\ \min(n, m + d)\mu & \text{if } t = (m + d, \max(n - 1, 0)). \end{cases} \tag{3}$$

Thus, the total transition rate per state $\Lambda(s, d)$ is simply the total of the possible transition rates:

$$\Lambda(s, d) \triangleq \lambda + \min(n, m + d). \tag{4}$$

The normalization rate for the SMDP is the maximum of $\Lambda(s, d)$, which we denote as follows:

$$\tilde{\Lambda} \triangleq \max \Lambda(s, d) = \max \left(\lambda + \min(n, m + d)\mu\right) = \lambda + M\mu. \tag{5}$$

With this knowledge, we can formally define the transition rates between states, which we defer to Appendix A. As for the objective function $C(s, d)$, this represents the cost to operate the cluster in state s, given decision d. The function consists of the following components:

(1) The cost to maintain each SU: $(m + d)C_S$.
(2) The surplus charges of unused time for (de)activated SUs: this triggers conditioned on a transition occurring *and* scaling occurring and is therefore $\Lambda(s, d)\mathbb{1}_{d\neq 0}C_D/\tilde{\Lambda}$.
(3) The SLA penalty of dropped requests: this is conditioned on a full buffer and the probability of an arrival, and thus equals $\lambda\mathbb{1}_{(n=N)}C_R/\tilde{\Lambda}$.
(4) The SLA penalty for stale requests: this applies to requests that have been in the queue beyond a particular threshold. To estimate this, we apply Little's Law on the assumption that requests are spread evenly across the $m + d$ SUs, and then apply the penalty based on the number of requests exceeding the threshold, resulting in $(n - \lambda(m + d)W)\mathbb{1}_{\frac{n}{(m+d)\lambda} > W}C_P$.

Param.	Value(s)	Param.	Model A	Model B	Model C
λ	50 req/hr	N	$\{100, 200, 400\}$	100	100
K	$\{5, 10, 20\}$	M	$\{12, 16, 32, 64\}$	6	3
W	$\{5, 10, 15\}$ mins	μ	5 req/hr	10 req/hr	20 req/hr
C_D	$\{0, 0.017 \times C_S, C_S\}$ kWh	C_S	0.01 kWh	0.02 kWh	0.04 kWh
C_A	2 kWh	C_P	0.075 kWh	0.134 kWh	0.581 kWh
		C_R	$\{7.47, 14.94, 29.88\}$ kWh	13.41 kWh	58.09 kWh

Table 3. Parameters governing the Kubernetes cluster in our numerical examples.

However, the SU cost (1) and stale request SLA penalties (4) are continuous costs and must be normalized when applied to the SMDP space. This results in the following definition for $C(s, d)$:

$$C(s,d) = \Big((m+d)C_S + \Lambda(s,d)\mathbb{1}_{d\neq 0}C_D + \lambda\mathbb{1}_{n=N}C_R + (n - \lambda(m+d)W)\mathbb{1}_{\frac{n}{(m+d)\lambda}>W}C_P\Big)\Big/\tilde{\Lambda}. \tag{6}$$

4.2.2 Attacker Stage MDP. The attacker SMDP is defined using a similar process, under the assumption that the defender's solution is already known. Given defender policy π, we let $\mathcal{S}^\pi \subset \mathcal{S}$ be the set of states the cluster occupies following the autoscaling policy. Then given the attacker decision space $\mathcal{A} = \{0, 1\}$ and attack power K as defined in Definition 3.1, the updated transition rates for states $s \longrightarrow t \in \mathcal{S}^\pi$ depend on whether an attack is active:

$$\begin{cases} \lambda & \text{if } t = (m+\pi(s), \min(n+1, 0)), \text{and } a = 0; \\ (K+1)\lambda & \text{if } t = (m+\pi(s), \min(n+1, 0)), \text{and } a = 1; \\ \min(n, m+\pi(s))\mu & \text{if } t = (m+\pi(s), \max(n-1, 0)). \end{cases} \tag{7}$$

This dependency affects the per-state transition rate and normalization rate:

$$\Lambda_a(s,a) \triangleq (K\mathbb{1}_{a=1}+1)\lambda + \min(n, m+\pi(s)). \tag{8}$$

$$\tilde{\Lambda}_a \triangleq \max \Lambda_a(s,a) = \max\left(\lambda + K\lambda\mathbb{1}_{a=1} + min(n, m+\pi(s))\mu\right) = (K+1)\lambda + M\mu. \tag{9}$$

Appendix B details the corresponding transition probabilities. As for the reward function $\mathcal{R}(s, a)$, the attacker seeks to maximize the cost to the defender, while minimizing the per-state cost of launching the attack. The former is essentially $C(s, \pi(s))$; however, the costs must be re-normalized in terms of the new $\tilde{\Lambda}_a$, and the arrival probabilities must be updated to account for the attacker's potential presence. Attack costs are defined such that C_A is proportional to attack power - as traffic is continuous, costs must also be normalized, resulting in $KC_A/\tilde{\Lambda}_a$, and a net stage reward of:

$$\begin{aligned}\mathcal{R}(s,a) =& \Big((m+\pi(s))C_S + \Lambda_a(s,a)\mathbb{1}_{\pi(s)\neq 0}C_D + (1+K\mathbb{1}_{a=1})\lambda\mathbb{1}_{n=N}C_R \\ &+ \left(n - (m+\pi(s))(K\mathbb{1}_{a=1}+1)\lambda W\right)\mathbb{1}_{\frac{n}{(m+\pi(s))(K\mathbb{1}_{a=1}+1)\lambda}>W}C_P - K\mathbb{1}_{a=1}C_A\Big)\Big/\tilde{\Lambda}_a.\end{aligned} \tag{10}$$

4.3 Numerical Examples

We evaluate our SMDP model by considering sample configurations derived from the cost-aware scheduling scenarios in [68]. The ranges for the parameters considered in these setups are listed in Table 3. In conducting these numeric evaluations, we are especially interested in:

(1) How attack power changes impact attacker behavior, given differences in relative costs, and;
(2) How the cloud provider billing structure, specifically implementation of minimum charges for machine activations, impacts incentives of where to attack or not attack.

We consider energy (in kWh) as a universal currency, as specific costs vary by jurisdiction and peak vs off-peak rates. Leveraging data from [29, 42], we consider three Amazon EC2 instance cluster Models, each featuring a baseline customer arrival rate of $\lambda = 50$ req/hr, a service rate of $\mu = 5$ req/hr per vCore, and default buffer size of $N = 100$.

A: $M = 12$ t2.micro instances (1 vCore, 1 GB RAM). An instance requires 10 watts of power to operate, and a request consumes approximately 75 watts to process. Thus, we let $C_S = 0.01$ kWh and $C_P = 0.075$ kWh. For this Model, in addition to the default SU, buffer configuration, we consider configurations $(M, N) \in \{(16, 100), (32, 200), (64, 400)\}$.

B: $M = 6$ t3.small instances (2 vCore, 2 GB RAM). Each machine requires 20 watts of power to operate, with requests consuming approximately 130 watts ($C_S = 0.02$ kWh, $C_P = 0.134$ kWh). Doubling the cores also doubles the service rate to $\mu = 10$ req/hr.

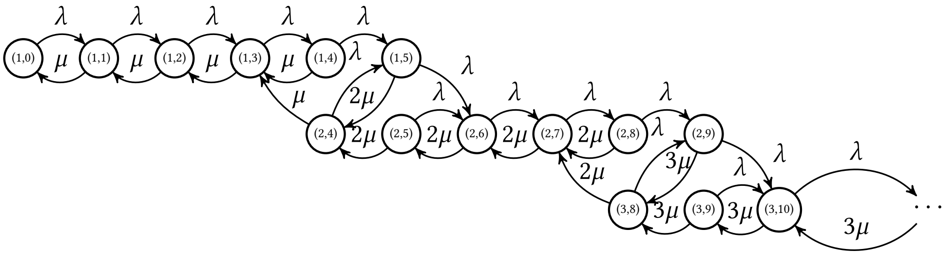
C: $M = 3$ al.xlarge instances (4 vCore, 8 GB RAM). Each machine requires 40 watts of power to operate and requests consume approximately 580 watts of power ($C_S = 0.04$ kWh, $C_P = 0.581$ kWh). The corresponding service rate is $\mu = 20$ req/hr.

The logic behind basing C_P on the cost to process requests is the notion that delayed requests represent electricity wasted. We consider varying thresholds in the set $W \in \{5, 10, 15\}$ mins to evaluate the impact of the SLA penalty on the scaling decision and attack incentives. By extension, as rejected requests occur when the request buffer is full, we let $C_R = NC_P$, resulting in $C_R \in \{7.47, 14.94, 29.88\}$ for Model A with buffer sizes $N \in \{100, 200, 400\}$; $C_R = 13.41$ for Model B; and $C_R = 58.09$ for Model C. The cost C_D, corresponding to minimum charges tied to SU operations, we let vary in the set $\{0, 0.017C_S, C_S\}$, representing different billing model variants: per-second/"actual usage" billing, per-minute billing, and per-hour based models, respectively.

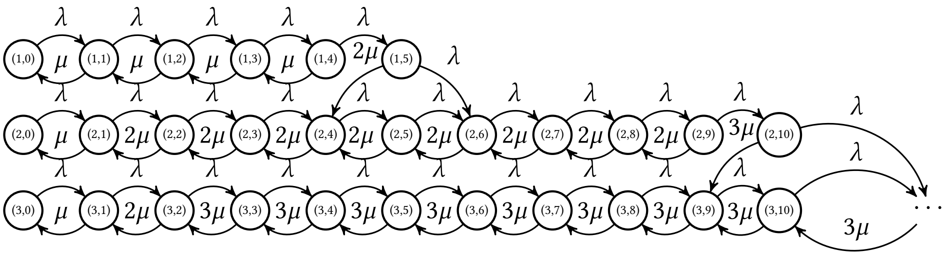
The attacker's cost is the energy requirements of traffic generation. For each unit of attack power, we assume the attacker operates a platform running a 2000W power supply for a cost of $C_A = 2$ kWh per unit K of attack power; such units are commercially available, and are marketed for use by crypto-miners [51]. Thus, it is reasonable to assume that the attacker has access to such a power supply within their attack generation platform. We allow the attack power K to vary in the set $\{5, 10, 20\}$, evaluating how varying levels of attacker traffic impact the relative reward gain against increased cost of additional traffic. These values are also chosen to remain below the level $K = 40$ utilized in prior works resulting in the Yo-Yo scenario in Figure 1 [18], to model an attacker attempting to transmit bursts at levels more likely to be confused for heavy customer traffic. Leveraging the Marmote [32] package to generate our solutions, we compute Discounted Value Iteration policies with discount factor $\beta = 0.95$ (favoring patient, long-term benefits) for each Model scenario according to the processes outlined in Sections 4.2.1- 4.2.2.

4.3.1 Autoscaling policies. We find that the greatest influence on the scaling policies comes from the billing mode, *i.e.* the granularity of the minimum charges as determined by C_D. We plot an example in Figure 2 for the Model A configuration with $M = 12, N = 100$ and $W = 5$ min SLA threshold. When $C_D = 0$ and no minimum charges are imposed for SU activations, the optimal policy is a hysteresis per Definition 3.2. Most critically, at each level, there is a threshold at which the cluster will scale down and remove SUs rather than keep resources idling that are not needed. While not shown in the figure, this also holds for the per-minute model ($C_D = 0.017C_S$). While minimum charges exist under such this model, the minimums are relatively small and thus do not create incentives to keep high levels of SUs active even at low traffic.

Conversely, when per-hour billing scenarios are in effect and $C_D = C_S$, the hysteresis hypothesis is violated and scale-down thresholds are removed. Because of how billing is applied in such an environment, the application is charged once an SU is active at any point during an hour, regardless of whether it remains active for the full hour. As a result, once active there is no incentive to deactivate a Service Unit, and it is more beneficial to leave SUs active with lower numbers of requests rather than scaling down unneeded SUs, in order to maintain a higher service rate and

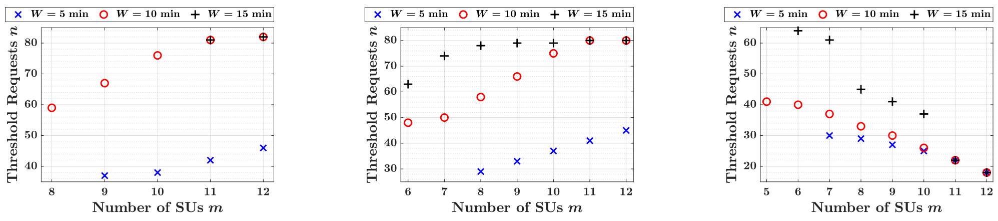


(a) $C_D = 0$



(b) $C_D = C_S$

Fig. 2. A subset ($1 \leq m \leq 3$) of the corresponding Markov chain solutions for a Model A configuration, given the parameters in Table 3 with $M = 12, N = 100$ and a wait time threshold of $W = 5$ min. When no minimum charges are applied ($C_D = 0$), the hysteresis hypothesis is satisfied. Service Units are activated and deactivated as necessary based on the number of requests present. If minimum per-hour charges are applied ($C_D = C_S$), incentives to shut down unneeded SUs are removed as charges are applied even when idle. This results in hysteresis being violated, and can be exploited by the adversary in attack planning.



$C_D = 0$ $C_D = 0.017 \times C_S$ $C_D = C_S$

Fig. 3. Attack thresholds for the Model A, $M = 12, N = 100$ configuration and attack power $K = 10$ given the parameters in Table 3. We see that raising the SLA threshold W for fixed values of C_D also raises the threshold at a given SU level as a greater number of requests must be present before the SLA violations occur. Similarly, increasing C_D lowers attack thresholds for fixed W through increasing cluster operating costs in a given configuration, particularly when per-hour minimum charges are imposed ($C_D = C_S$).

decrease the probability of the request buffer reaching a level at which a threshold to scale up to further levels of SU service is reached. This behavior can be exploited by the attacker to craft their strategy of where to inject traffic.

Model Configuration		$C_D = 0$	$C_D = 0.017C_S$	$C_D = C_S$
$K = 5$	Model A $M = 12, N = 100$	$(11, 81)$	$(5, 62)$	$(5, 63)$
	Model A $M = 16, N = 100$	$(15, 88)$	$(7, 82)$	$(7, 77)$
	Model A $M = 32, N = 200$	$(32, 195)$	$(23, 193)$	$(19, 194)$
	Model A $M = 64, N = 400$	$(63, 397)$	$(62, 397)$	$(54, 398)$
	Model B	$(4, 38)$	$(3, 28)$	$(3, 27)$
	Model C	$(2, 13)$	$(1, 6)$	$(1, 6)$
$K = 10$	Model A $M = 12, N = 100$	$(11, 81)$	$(6, 63)$	$(6, 64)$
	Model A $M = 16, N = 100$	$(15, 88)$	$(7, 83)$	$(7, 80)$
	Model A $M = 32, N = 200$	$(32, 195)$	$(21, 192)$	$(19, 193)$
	Model A $M = 64, N = 400$	$(63, 397)$	$(62, 397)$	$(48, 397)$
	Model B	$(3, 35)$	$(2, 21)$	$(2, 21)$
	Model C	$(1, 4)$	*	*
$K = 20$	Model A $M = 12, N = 100$	$(11, 81)$	$(6, 70)$	$(6, 72)$
	Model A $M = 16, N = 100$	$(15, 88)$	$(8, 86)$	$(7, 87)$
	Model A $M = 32, N = 200$	$(32, 195)$	$(21, 192)$	$(20, 193)$
	Model A $M = 64, N = 400$	$(63, 397)$	$(62, 397)$	$(46, 397)$
	Model B	$(3, 35)$	$(2, 22)$	$(2, 22)$
	Model C	$(1, 1)$	*	*

Table 4. Threshold state (m, n) corresponding to the minimum SUs at which the attacker will initiate the attack decision under each cluster configuration, and minimum cost C_D combination for the $W = 15$ min SLA threshold scenario. The asterisks indicate scenarios in which it is universally beneficial to initiate an attack.

4.3.2 *Attacker behavior.*

Evaluating attacker behavior with respect to a given defender policy, we determine attacker strategies take the form of a threshold scenario. That is, for each level of SU scaling $m \in [1, M]$ one of the following will hold:

- For all $(m, n) \in \mathcal{S}^\pi$ the attacker chooses $a = 0$ (*i.e.*, never attacks);
- There exists n_m where for $n \geq n_m$ and $(m, n) \in \mathcal{S}^\pi$ the attacker chooses $a = 1$ and attacks.

In Figure 3 we plot the thresholds at each level resulting under the Model A, $M = 12, N = 100$ configuration for each C_D value and attack power $K = 10$ ($K \in \{5, 20\}$ are plotted in Figure 7 in Appendix D). Comparing the plots, several trends emerge. First, imposing minimum charges $C_D > 0$ generally lowers the thresholds where attacks are initiated. This results from increased costs of cluster operation in a given configuration, which in turn increases attacker rewards when comparing across fixed attack power K (and therefore corresponding fixed stage costs KC_A). In particular, when per-hour minimum charges are implemented ($C_D = C_S$), the hysteresis violation results in the request thresholds *decreasing* as a function of the SU level for a given combination of attack power and SLA threshold. The tight constraint on the request buffer ($N = 100$) relative to the rate of adversarial arrivals ($\lambda_a = 500$ req/hr) increases the probability of reaching the full buffer state. Similar observations will hold in the $K = 5$ ($\lambda_a = 250$ req/hr) and $K = 20$ ($\lambda_a = 1000$ req/hr) cases. Second, increasing the SLA penalty threshold W raises the minimum threshold for attack for fixed C_D and attack power combinations, as the decreased charges to the defender lowers the attacker reward and therefore their incentive. Third, while increasing K increases the rate of traffic, the corresponding increase to KC_A results in an offset to the benefit received. As a result, the thresholds for attack at $K = 10$ and $K = 20$ are similar for fixed SLA thresholds and C_D, suggesting that there is an upper bound on the benefit of increasing the rate of attacker traffic.

Building from this observation, we compare the thresholds corresponding to the minimum number of SUs where the attacker opts to attack across all three Model configurations, and multiple variations of SU capacity and request buffer sizes for Model A. The thresholds for the $W = 15$ mins SLA threshold scenario are noted in Table 4. The remaining scenarios are contained in Table 9 in Appendix D. Comparing Models B and C, with fewer but more powerful SUs, to the default Model A configuration, the decreased SU capacity results in increased incentive to launch attacks relative to available SU levels despite the increased service rates. In fact, imposition of minimum

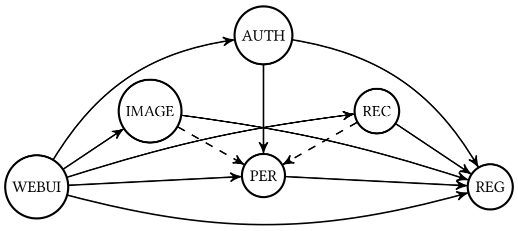


Fig. 4. Architecture of the TeaStore application web service components: WebUI, Authentication (AUTH), Persistence (PER), Recommender (REC), Image, and Registry (REG). Solid lines indicate dependencies between components affecting application scaling. Dashed lines indicate interactions occurring once at initial startup.

charges under Model C results in situations where it is universally beneficial to launch an attack for $K > 5$. Conversely, comparing varying SU and request buffer capacities under Model A: $(M, N) \in \{(12, 100), (16, 100), (32, 200), (64, 400)\}$, we find that increasing the capacity results in the relative threshold being raised to a minimum of two-thirds of available SUs in the $M \in \{32, 64\}$ cases with request buffers near the maximum capacity. When no minimum charges are imposed ($C_D = 0$), in these cases the minimum SUs increases to at worst one less than the maximum level, with a request buffer 98% full. These results imply that system configuration heavily dictates attack incentive, and that resource-constrained systems are especially vulnerable due to operating under lean conditions, while extending the available capacity pool for the same configuration significantly decreases attack incentive. Additionally, this reinforces the influence billing configuration has on attack incentive, as real-time billing structures in particular disincentivize attacks more stringently than structures imposing minimum charges.

5 EXPERIMENTAL EVALUATIONS

In this section we conduct experimental evaluations against a realistic Kubernetes cluster. From our model observations, our goals with the experiments are to:

(1) Quantify the impacts of billing methods on the attack incentive, and
(2) Quantify the impacts of customer and attacker patterns on the resource constraints and the impacts of varying attacker strengths riding on top of customer traffic patterns.

5.1 Experimental Setup

The experiments were run within a Docker [19] container hosted on a platform running a 13th Generation Intel Core i9-13900K processor with 32 total logical threads, and 64 GB of memory available. The Docker container runs a Kubernetes cluster consisting of the TeaStore microservices testbed [71], an e-commerce application emulator comprising the following components:

- WebUI - web application serving as the platform users directly interface with.
- Image - image provider service supplying images associated with the product catalog.
- Persistence - stores order history; supports registered users viewing individual order history.
- Authentication - authenticates user credentials.
- Recommender - suggests products to users based on current selection.
- Registry - associates newly-created service instances to the cluster.

Interactions between components are mapped in Figure 4. WebUI interacts with all other components, as it is the platform utilized to navigate the application as a whole. On the other hand, the Recommender only needs to generate recommendations on initial startup (as recommendations are static and not user-specific), which can then be cached by Persistence instances rather than be run

Component	WEBUI	AUTH	PER	REC	IMAGE	REG
CPU mCores	1200	900	900	800	1100	1200
Memory (MB)	1024	1024	1024	1024	1024	1024

Table 5. Configuration settings for each instance of the TeaStore components: WebUI, Authentication (AUTH), Persistence (PER), Recommender (REC), Image, and Registry (REG).

constantly. These differing requirements of effort, as well as the varying resource requirements as noted in Table 5, once again necessitates a common Service Unit definition - we use a definition of $\text{SU} = \sup(\lceil \text{CPU cores} \rceil, \lceil (\text{RAM GB})/4 \rceil)$ in use by the New England Research Cloud [50]. To emulate a user erring on the side of avoiding SLA violations [66], the components are set to scale at 80% average CPU utilization.

The tests were run using a modified version of the TeaStore Test Plan script [71], utilizing Apache jMeter [9] to feed simultaneous streams of legitimate and attacker traffic. We utilize Prometheus [55] and Grafana [26] to monitor and visualize the traffic and the resulting number of SUs generated, respectively. Each test we consider was run over a two-hour period, during which the first hour consisted exclusively of legitimate traffic. Customer requests are synthetically generated at an overall rate of $\lambda = 36$ req/sec, derived from studies of a shared cluster scenario with a baseline rate of approximately $11,000$ req/12 hr [43], which we apply to our platform to emulate a multi-cluster system (with 1 physical core = 1 cluster), and a 10× traffic pattern to emulate a larger than typical spike driven by special event demand. We consider three types of customer traffic patterns, corresponding to varying workload request patterns observed in real systems [36]:

(1) An *On/Off* pattern of traffic alternating between arrival periods of 8 minutes in duration and no traffic for a 2-minute period.
(2) A *Bursty* pattern where traffic is subject to random bursts - customers are grouped into 50 threads with a 500ms delay between threads using jMeter's Precise Throughput timer which generates traffic within each thread according to an independent process.
(3) A *Random* pattern where traffic arrives according to a continuous Poisson random distribution.

Our adversary in these scenarios transmits traffic during the second hour. The adversary does not have a defined, fixed monetary budget; *however*, the attacker *is* constrained in terms of time, as the window of attack is over the course of a single hour period. As under our model, we consider an attacker transmitting with attack powers $K \in \{5, 10, 20\}$. In practice, we observe that most cloud services utilize a variation of per-minute or per-hour charges [7, 24, 50]. This informs the attacker strategy, wherein the attacker launches attacks in two-minute bursts whenever the number of Service Units is determined to be decreasing, and the current number of Service Units is at least 5. As seen in the MDP model, within constrained environments the attacker will generally benefit when attacking if the number of SUs exceeds the mean, and the measured mean utilization at baseline under a random traffic load is 4. Thus, the attacker chooses 5 as the target and will launch bursts with the objective of keeping the cluster in a configuration which exceeds this state. Unlike the strictly periodic Yo-Yo however, the attacker has a sense of the current state and is not attacking on a strict timer. We evaluate the impact of the attack against various billing modes to compare the relative attack efficiency per Definition 3.3:

- Per-minute - usage charged per minute based on current minute's measured utilization. Over the course of an hour, given SU_t as the current utilization at minute t, this is equal to $\sum_{t=0}^{60} \frac{1}{60} SU_t$. This is the mode the cluster is assumed to be utilizing in practice.
- Per-hour - usage charged per hour based on maximum utilization during the last hour. This is simply $\max_{t \in [0,60]} SU_t$. However, the attacker would have incentive to behave differently

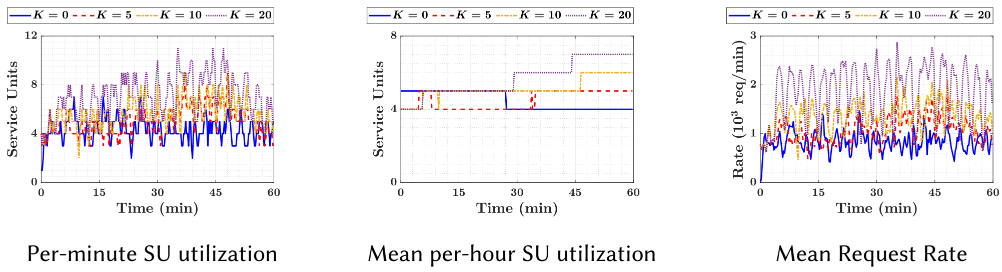


Fig. 5. The Service Rate utilization per-minute and mean per-hour, and the corresponding request rates for each attack strength compared to the baseline utilization ($K = 0$) under Random customer traffic. Measuring utilization based on (rolling) mean per-hour smooths over fluctuations caused by attacker traffic to an extent.

Traffic Type	Billing Mode	Baseline	$K = 5$	$K = 10$	$K = 20$
Bursty	Per-Minute	4.1958	4.5167	6.2917	11.025
	Per-Hour (Max)	7	6	11	18
	Per-Hour (Mean)	4.4542	4.4792	5.1250	5.6792
Random	Per-Minute	4.3042	4.9542	5.6708	7.0667
	Per-Hour (Max)	7	9	9	11
	Per-Hour (Mean)	4.8958	4.3125	4.7798	7.5292

Table 6. The total number of computed Service Units charged under each billing scenario under consideration. On-Off is omitted as the Baseline utilization is always under 5 SU and attacks will not trigger.

if all that was required was to force a single large burst at some point during the 60 minute period. Thus, this is primarily meant for illustrative purposes to demonstrate the maximum SUs created at peak.

- Per-hour mean - usage charged per hour based on mean utilization rate. The mean is computed as a rolling average, and thus the charge is the weighted sum of the mean computed per-minute: $\sum_{t=0}^{60} \frac{1}{60}\overline{SU}_t$.

5.2 Rate of Service Unit Utilization

Evaluating the Service Unit utilization for each customer traffic type and billing mode configuration, we find that when viewing at per-minute granularity, scaling occurs frequently in response to traffic variations even under baseline load. By comparison, when viewing the mean per-hour utilization rate, by its nature as a rolling average such volatilities are smoothed over. This is visualized in Figure 5 for the Random traffic type, and in Figure 8 in Appendix D for the Bursty traffic type. Additionally, charging based on mean utilization rate vs. the real-time utilization, or imposing a minimum charge for maximum SU utilization, reduces incentives to attack, as generating higher amounts of traffic does not necessarily translate to equivalent charges imposed on the target. We show this in Table 6 via direct comparison of the SUs created under each scenario.

Under Bursty traffic, per-hour mean billing generates $\{1.01, 1.15, 1.28\}$ SUs times the baseline charge for attack power $K \in \{5, 10, 20\}$, compared to $\{1.08, 1.5, 2.63\}$ times baseline under per-minute billing or $\{0.86, 1.57, 2.57\}$ times baseline under maximum utilization charges. In this, we see also that the underlying type of traffic influences the impact of the attack, as under random traffic the per-hour mean scenario generates $\{0.88, 0.98, 1.54\}$ SUs times baseline under each attack,

Traffic Type	Attack Power	$frac_{attack}$	Per-Min	Per-Hr (Max)	Per-Hr (Mean)
Bursty	$K = 5$	0.3	0.214	-0.667	0.017
	$K = 10$	0.43	0.487	*0.930*	*0.156*
	$K = 20$	0.6	*0.569*	0.917	0.102
Random	$K = 5$	0.37	*0.351*	*1.08*	-0.315
	$K = 10$	0.63	0.217	0.318	-0.018
	$K = 20$	0.67	0.207	0.299	*0.197*

Table 7. Attack efficiency for each attack power scenario and traffic type-billing model combination. Italicized values indicate the most efficient attack for the given scenario.

but the corresponding per-minute and per-hour max ratios ($\{1.15, 1.32, 1.64\}$ and $\{1.29, 1.29, 1.57\}$) are lower for higher attack powers. This indicates that the greater levels of attacker traffic are able to take advantage of the large waves of customer traffic to generate requests, compared to a randomized pattern where the customer traffic does not correspond to a discrete pattern; while the attacker bursts are an alternating on-off but without a set idle period.

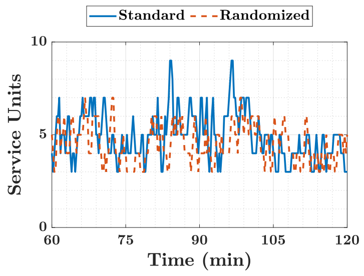
Further, the large amount of Service Units generated in the $K = 20$ case are offset by the large spikes in request rates, which are detectable when plotted against the baseline patterns, indicating an increased chance of triggering a filter flagging for anomalous traffic and offsetting any benefits which might be gained. At the other extreme, the customer on-off utilization pattern has a per-minute utilization at baseline of 3.27 SUs, and as seen in Figure 9 of Appendix D, the SU utilization rate does not exceed 4 SUs. Thus, under this scenario the attack will not trigger. In the next section, we consider the efficiency of each attack under the other two traffic scenarios relative to traffic generation.

5.3 Relative Economic Damage per Billing Mode

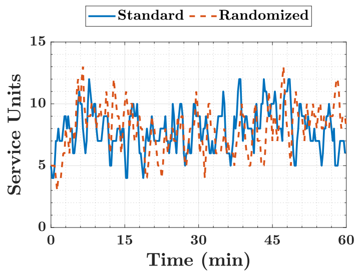
To evaluate the efficacy of each attack, in Table 7 we apply the efficiency definition from Definition 3.3 and compare the difference in SUs created by the attack compared to the measured baseline against the relative power of the attack for each customer traffic type and billing mode. While rates of request failures are hypothetically relevant, we determine that the actual count of request failures is negligible and therefore are a non-factor in this case. The fraction of time spent attacking $frac_{attack}$ is computed per traffic type. In the Random customer traffic scenario, attack bursts are initiated for $\{11, 19, 20\}$ cycles respectively for $K \in \{5, 10, 20\}$. Given 2-minute cycles, this yields $frac_{attack} \in \{0.37, 0, 63, 0.67\}$ over the course of the hour period. Similarly, for the Bursty traffic scenario, the respective number of attack cycles are $\{9, 13, 18\}$ for $frac_{attack} \in \{0.3, 0.43, 0.6\}$.

In the Bursty case we find that the $K = 10$ attack is generally the most efficient. While the raw traffic sent in the $K = 20$ case results in a greater number of SUs created overall, which is reflected in $K = 20$ being the most effective under per-minute billing, under per-hour maximum or mean charges the increased costs of attack do not result in significantly greater SU generation. This is compounded by the nature of the adversary, who while attacking in a state-aware fashion creates a feedback loop where the measured state of the system is the result of previous attacks, and a decrease from such a state is cause to resume attacking, resulting in behavior closer to traditional DDoS. This is supported by the unconstrained MDP model. Once crossing the threshold to initiate the attack, the only incentive to stop is if enough departures occur for the request buffer queue size to drop, an eventuality made less likely by sending additional traffic into the system.

The Random case further reinforces however that, generally, there are limitations to the efficacy of increasing traffic. Here, the $K = 5$ is the more efficient attack except in the per-hour mean case, and this is primarily due to the baseline having generated a higher amount of SUs than the attack



Baseline



$K = 20$

Fig. 6. Comparison of Service Unit utilization with and without scaling randomization in place under baseline utilization and attack power $K = 20$. Randomization has a smoothing effect on scaling, resulting in comparable utilization with less aggressive scaling-down under baseline. However, sufficient traffic overwhelming scaling is a potential drawback as seen with the attack having similar utilization curves under both scenarios.

when computing over the rolling average period. In general however, the additional traffic under attack powers $K \in \{10, 20\}$ do not generate sufficient SUs to outweigh the extra cost, unlike in the Bursty case where the the combined bursts of customer and attacker traffic push the system into higher states rendering a $K = 10$ attack more effective.

6 COUNTERMEASURES

As noted previously, there are many existing potential countermeasures to EDoS attacks, most deriving from Machine Learning techniques. However, the main drawbacks of such techniques are the resources required to train underlying models, as well as the expectation that training data utilized is representative of the intended output. Thus, we propose a randomization method inspired by rate adaptation algorithms [54] to minimally inconvenience normal workloads while disrupting attempts to attack the system. To do so here, we redirect the autoscaler to a routine running a random number generator via the HPA custom scaling metrics capability [40]. If the generator returns true, then the actual state of the system is returned and scaling is handled accordingly. Otherwise, the current auto-scale cycle is skipped. Such an approach is a potential means to employ a simplified defense mechanism against adversaries launching attacks on the assumption that current state knowledge is sufficient to determine whether scaling is likely to occur. A fully comprehensive solution requires fine tuning of the parameters necessary for scaling and optimization of the randomization factor. For our experiment the RNG is weighted evenly in terms of scaling vs. not scaling, against a Random customer traffic workload. The customer traffic and attack are generated in similar fashion to the experiments in Section 5.

6.1 Empirical Study of Randomized Defense

We consider a per-minute utilization model for the charges. Computing the corresponding costs and efficiencies in Table 8, we find that for $K < 20$, we indeed end up with fewer SUs compared to standard scaling, benefiting the application under attack conditions through reduced charges. While the relative attack efficiency *increases*, this is because efficiency is being compared to the comparable baseline, and baseline charges under randomization are 90.6% compared to standard HPA; thus, the application is indeed better off. However, in terms of actual SU charges, when $K = 20$ the application is worse off, although only by slightly more than 1%. This indicates that sufficiently

Charges	Baseline	$K = 5$	$K = 10$	$K = 20$
Standard HPA	4.866	5.517	6.164	7.915
Randomized HPA	4.408	5.105	6.07	8
Ratio Random/Standard	0.906	0.925	0.984	1.011

Attack Efficiency	$K = 5$	$K = 10$	$K = 20$
Standard HPA	*0.228*	0.185	0.191
Random HPA	*0.247*	0.237	0.225
Ratio Random/Standard	1.083	*1.281*	1.178

Table 8. Service Units charged and corresponding Attack Efficiencies under per-minute billing under standard vs randomized HPA autoscaling scenarios. In general, fewer SUs are created under randomization, and the $K = 5$ attack strength will have the greatest relative efficiency, although $K = 10$ sees the greatest relative efficiency increase.

large bursts of traffic can still potentially overwhelm the randomization. Thus, while the results indicate a proof of concept, more work is necessary to identify optimal parameter ranges for the randomization process.

7 CONCLUSIONS

In this work, we introduced a Stackelberg Security Game leveraging Markov Decision Process-based models for determining when adversarial users should inject traffic into Kubernetes clusters given various factors, including costs related to per-resource minimum billing charges. We find that generally, imposing such charges will increase incentives to attack with Economic Denial of Sustainability objectives at lower thresholds, especially in resource-constrained environments where it is easier to trigger Service Level Agreement violation penalties. Conversely, usage-based billing models in clusters with sufficient capacities disincentivize attacks unless the system is already at or very close to the saturation point of available resources. Further, we find that generally there are limits to the benefits of increased traffic, as the thresholds do not increase for higher attack powers due to increased costs. Performing experiments on a Kubernetes cluster, this is reinforced. While the type of customer traffic and billing mode influences the level of Service Units ultimately generated, generally lower attack powers are more efficient even if higher levels of traffic result in a higher raw Service Unit generation. Randomization of autoscaling has the potential to mitigate the attack impacts by lowering the number of Service Units generated.

One drawback of the model is that the optimizations we derived are global in nature and do not consider a finite monetary attacker budget. To do so, however, requires incorporating temporal information regarding duration of attacks, which can render an MDP space intractable. Future work could address this by incorporating techniques (e.g., from [11]) to address the issue and enhance the model. Additionally, the attacker in our experiments created a feedback loop unto themselves, inspired by the incentives from the MDP model. To address this, we propose further experiments with additional configurations, including that of an attacker who evaluates over a sliding window of time and will not attack unless a minimum duration of time has passed. Additionally, we propose to evaluate randomization over additional traffic types, as well as a range of randomization weights, to evaluate the efficacy over a wider range of parameters.

ACKNOWLEDGMENTS

This work was supported in part by grants from the Red Hat Collaboratory at Boston University (2025-01-RH05) and the U.S. National Science Foundation (AST-2229104). Zaoxing Liu was supported in part by U.S. NSF grants SaTC-2415754 and CNS-2431093.

REFERENCES

[1] Chaitanya Agarwal, Shibashis Guha, Jan Křetínský, and M. Pazhamalai. 2024. PAC statistical model checking of mean payoff in discrete- and continuous-time MDP. *Formal Methods in System Design* (8 2024), 1–43. https://doi.org/10.1007/s10703-024-00463-0

[2] Silvia E. Albarran and Julio B. Clempner. 2019. A Stackelberg security Markov game based on partial information for strategic decision making against unexpected attacks. *Engineering Applications of Artificial Intelligence* 81 (5 2019), 408–419. https://doi.org/10.1016/j.engappai.2019.03.010

[3] Theyazn H. H. Aldhyani and Hasan Alkahtani. 2022. Artificial Intelligence Algorithm-Based Economic Denial of Sustainability Attack Detection Systems: Cloud Computing Environments. *Sensors* 22 (6 2022), 4685. Issue 13. https://doi.org/10.3390/s22134685

[4] Laha Ale, Ning Zhang, Scott A. King, and Jose Guardiola. 2021. Spatio-temporal Bayesian Learning for Mobile Edge Computing Resource Planning in Smart Cities. *ACM Transactions on Internet Technology* 21 (8 2021), 1–21. Issue 3. https://doi.org/10.1145/3448613

[5] Arnold O. Allen. 2014. *Probability, Statistics, and Queueing Theory With Computer Science Applications* (2nd ed.). Academic Press, San Diego, CA, USA.

[6] Amazon Web Services. 2024. AnomalyMonitor - AWS Billing and Cost Management. https://docs.aws.amazon.com/aws-cost-management/latest/APIReference/API_AnomalyMonitor.html

[7] Amazon Web Services. 2024. EC2 On-Demand Instance Pricing. https://aws.amazon.com/ec2/pricing/on-demand/

[8] Amazon Web Services. 2024. Red Hat Enterprise Linux on AWS Pricing. https://aws.amazon.com/partners/redhat/rhel-pricing/

[9] Apache Software Foundation. 1999. Apache jMeter. https://jmeter.apache.org/

[10] Maria-Florina Balcan, Avrim Blum, Nika Haghtalab, and Ariel D. Procaccia. 2015. Commitment Without Regrets: Online Learning in Stackelberg Security Games. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation* (Portland, Oregon, USA). Association for Computing Machinery, New York, NY, USA, 61–78. https://doi.org/10.1145/2764468.2764478

[11] Johannes K Becker and David Starobinski. 2022. Optimizing Freshness in IoT Scans. In *2022 IEEE 8th World Forum on Internet of Things (WF-IoT)* (Yokohama, Japan). IEEE, New York, NY, USA, 1–6. https://doi.org/10.1109/WF-IoT54382.2022.10152289

[12] Anat Bremler-Barr, Eli Brosh, and Mor Sides. 2017. DDoS attack on cloud auto-scaling mechanisms. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications* (Atlanta, GA, USA). IEEE, New York, NY, USA, 1–9. https://doi.org/10.1109/INFOCOM.2017.8057010

[13] Anthony Byrne, Sadie L. Allen, Shripad Nadgowda, and Ayse K. Coskun. 2019. Praxi: Cloud Software Discovery That Learns From Practice. In *Proceedings of the 20th International Middleware Conference Demos and Posters* (Davis, CA, USA). Association for Computing Machinery, New York, NY, USA, 23–24. https://doi.org/10.1145/3366627.3368111

[14] Hao Chen, Sastry S. Duri, Vasanth Bala, Nilton T. Bila, Canturk Isci, and Ayse K. Coskun. 2014. Detecting and identifying system changes in the cloud via discovery by example. In *2014 IEEE International Conference on Big Data (Big Data)* (Washington, DC, USA). IEEE, New York, NY, USA, 90–99. https://doi.org/10.1109/BigData.2014.7004217

[15] Fahad Zaman Chowdhury, Laiha Binti Mat Kiah, M.A. Manazir Ahsan, and Mohd Yamani Idna Bin Idris. 2017. Economic denial of sustainability (EDoS) mitigation approaches in cloud: Analysis and open challenges. In *2017 International Conference on Electrical Engineering and Computer Science (ICECOS)* (Palembang, Indonesia). IEEE, New York, NY, USA, 206–211. https://doi.org/10.1109/ICECOS.2017.8167135

[16] CloudZero. 2024. Kubernetes Cost Optimization Software. https://www.cloudzero.com/solutions/kubernetes/

[17] Vincent Conitzer and Tuomas Sandholm. 2006. Computing the optimal strategy to commit to. In *Proceedings of the 7th ACM conference on Electronic commerce* (Ann Arbor, Michigan, USA). ACM, New York, NY, USA, 82–90. https://doi.org/10.1145/1134707.1134717

[18] Ronen Ben David. 2021. *Kubernetes Auto-Scaling: YoYo attack vulnerability and mitigation.* Ph. D. Dissertation. Reichman University.

[19] Docker Inc. 2025. Docker: Accelerated Container Application Development. https://www.docker.com/

[20] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. 2013. {ZMap}: Fast internet-wide scanning and its security applications. In *22nd USENIX Security Symposium (USENIX Security 13)* (Washington, DC, USA). USENIX Association, 605–620.

[21] A.A. Economides and J.A. Silvester. 1990. Priority load sharing: An approach using Stackelberg games. In *Proceedings of the 28th Annual Allerton Conference on Communications, Control and Computing*. 674–683.

[22] Rodrigo Ferroni, Dmitriy Novikov, Achraf Souk, Joanna Knox, Anuj Butail, and Harith Gaddamanugu. 2023. AWS Best Practices for DDoS Resiliency: AWS Whitepaper. (8 2023). https://docs.aws.amazon.com/whitepapers/latest/aws-best-practices-ddos-resiliency/aws-best-practices-ddos-resiliency.html

[23] Gartner. 2024. Gartner Forecasts Worldwide Public Cloud End-User Spending to Total $723 Billion in 2025. https://www.gartner.com/en/newsroom/press-releases/2024-11-19-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-total-723-billion-dollars-in-2025

[24] Google Cloud. 2024. Google Kubertes Engine (GKE). https://cloud.google.com/kubernetes-engine/#pricing

[25] Hima Govind and Horacio GonzaleznVelez. 2021. Benchmarking Serverless Workloads on Kubernetes. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)* (Melbourne, Australia). IEEE, New York, NY, USA, 704–712. https://doi.org/10.1109/CCGrid51090.2021.00085

[26] Grafana Labs. 2015. Grafana. https://grafana.com/

[27] Daniel Guerrero, Alin A. Carsteanu, and Julio B. Clempner. 2018. Solving Stackelberg security Markov games employing the bargaining Nash approach: Convergence analysis. *Computers & Security* 74 (5 2018), 240–257. https://doi.org/10.1016/j.cose.2018.01.005

[28] Daniel Guerrero, Alin A. Carsteanu, and Julio B. Clempner. 2018. Solving Stackelberg security Markov games employing the bargaining Nash approach: Convergence analysis. *Computers & Security* 74 (5 2018), 240–257. https://doi.org/10.1016/j.cose.2018.01.005

[29] David Guyon, Anne Cécile Orgerie, Christine Morin, and Deb Agarwal. 2019. Involving users in energy conservation: a case study in scientific clouds. *International Journal of Grid and Utility Computing* 10 (2019), 272. Issue 3. https://doi.org/10.1504/IJGUC.2019.099667

[30] Md. Sharafat Hossain, Md. Alamgir Hossain, and Md. Saiful Islam. 2024. I-MPaFS: enhancing EDoS attack detection in cloud computing through a data-driven approach. *Journal of Cloud Computing* 13 (10 2024), 151. Issue 1. https://doi.org/10.1186/s13677-024-00699-5

[31] Imperva. 2021. Global DDoS Threat Landscape Report. (2021). https://www.imperva.com/resources/reports/Imperva_2021-DDoS-Report.pdf

[32] Alain Jean-Marie and Emmanuel Hyon. 2024. Marmote's documentation. https://marmote.gitlabpages.inria.fr/marmote/index.html

[33] Wei Jiang, Bin Han, Mohammad Asif Habibi, and Hans Dieter Schotten. 2021. The Road Towards 6G: A Comprehensive Survey. *IEEE Open Journal of the Communications Society* 2 (2021), 334–366. https://doi.org/10.1109/OJCOMS.2021.3057679

[34] Li Ju, Prashant Singh, and Salman Toor. 2021. Proactive autoscaling for edge computing systems with kubernetes. In *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion* (Leicester, United Kingdom). Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3492323.3495588

[35] Debarun Kar, Thanh H. Nguyen, Fei Fang, Matthew Brown, Arunesh Sinha, Milind Tambe, and Albert Xin Jiang. 2018. *Trends and Applications in Stackelberg Security Games*. Springer International Publishing, Cham, 1223–1269. https://doi.org/10.1007/978-3-319-44374-4_27

[36] In Kee Kim, Wei Wang, Yanjun Qi, and Marty Humphrey. 2016. Empirical Evaluation of Workload Forecasting Techniques for Predictive Cloud Resource Scaling. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)* (San Francisco, CA, USA). IEEE, New York, NY, USA, 1–10. https://doi.org/10.1109/CLOUD.2016.0011

[37] In Kee Kim, Wei Wang, Yanjun Qi, and Marty Humphrey. 2022. Forecasting Cloud Application Workloads With *CloudInsight* for Predictive Resource Management. *IEEE Transactions on Cloud Computing* 10 (7 2022), 1848–1863. Issue 3. https://doi.org/10.1109/TCC.2020.2998017

[38] M. Yu. Kitaev and Richard F. Serfozo. 1999. M/M/1 Queues with Switching Costs and Hysteretic Optimal Control. *Operations Research* 47 (4 1999), 310–312. Issue 2. https://doi.org/10.1287/opre.47.2.310

[39] The Kubernetes Authors. 2025. Cluster Autoscaling. https://kubernetes.io/docs/concepts/cluster-administration/cluster-autoscaling/

[40] The Kubernetes Authors. 2025. Horizontal Pod Autoscaling. https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

[41] The Kubernetes Authors. 2025. Kubernetes. https://kubernetes.io

[42] Mascha Kurpicz, Anne-Cecile Orgerie, and Anita Sobe. 2016. How Much Does a VM Cost? Energy-Proportional Accounting in VM-Based Environments. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)* (Heraklion, Greece). IEEE, New York, NY, USA, 651–658. https://doi.org/10.1109/PDP.2016.70

[43] Christopher Liao, Yonatan Klausner, David Starobinski, Eran Simhon, and Azer Bestavros. 2018. A case study of a shared/buy-in computing ecosystem. *Cluster Computing* 21 (9 2018), 1595–1606. Issue 3. https://doi.org/10.1007/s10586-018-2256-2

[44] J. Longo, E. De Mulder, D. Page, and M. Tunstall. 2015. *SoC It to EM: ElectroMagnetic Side-Channel Attacks on a Complex System-on-Chip*. Springer, Berlin, Heidelberg, 620–640. https://doi.org/10.1007/978-3-662-48324-4_31

[45] John C.S. Lui and Leana Golubchik. 1999. Stochastic complement analysis of multi-server threshold queues with hysteresis. *Performance Evaluation* 35 (3 1999), 19–48. Issue 1-2. https://doi.org/10.1016/S0166-5316(98)00043-1

[46] Subrota Kumar Mondal, Xiaohai Wu, Hussain Mohammed Dipu Kabir, Hong-Ning Dai, Kan Ni, Honggang Yuan, and Ting Wang. 2023. Toward Optimal Load Prediction and Customizable Autoscaling Scheme for Kubernetes. *Mathematics* 11 (6 2023), 2675. Issue 12. https://doi.org/10.3390/math11122675

[47] Marco Antonio Sotelo Monge, Jorge Maestre Vidal, and Gregorio Martínez Pérez. 2019. Detection of economic denial of sustainability (EDoS) threats in self-organizing networks. *Computer Communications* 145 (9 2019), 284–308. https://doi.org/10.1016/j.comcom.2019.07.002

[48] Vahid Mirzaebrahim Mostofi, Evan Krul, Diwakar Krishnamurthy, and Martin Arlitt. 2023. Trace-Driven Scaling of Microservice Applications. *IEEE Access* 11 (2023), 29360–29379. https://doi.org/10.1109/ACCESS.2023.3260069

[49] Marco A.S. Netto, Carlos Cardonha, Renato L.F. Cunha, and Marcos D. Assuncao. 2014. Evaluating Auto-scaling Strategies for Cloud Computing Environments. In *2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems* (Paris, France). IEEE, New York, NY, USA, 187–196. https://doi.org/10.1109/MASCOTS.2014.32

[50] New England Research Cloud. 2024. How Does NERC Pricing Work? https://nerc-project.github.io/nerc-docs/get-started/cost-billing/how-pricing-works/

[51] NewEgg. 2025. 2000W Bitcoin mining PSU PC Power Supply Computer Mining Rig 8 GPU ATX Ethereum Coin 12v 4 pin power supply. https://www.newegg.com/p/1HU-034G-00005

[52] Abraham Neyman. 2017. Continuous-time stochastic games. *Games and Economic Behavior* 104 (7 2017), 92–130. https://doi.org/10.1016/j.geb.2017.02.004

[53] Joseph Noor, Ahmed Ali-Eldin, Luis Garcia, Chirag Rao, Venkat R. Dasari, Deepak Ganesan, Brian Jalaian, Prashant Shenoy, and Mani Srivastava. 2019. The Case for Robust Adaptation: Autonomic Resource Management is a Vulnerability. In *MILCOM 2019 - 2019 IEEE Military Communications Conference (MILCOM)* (Norfolk, VA, USA). IEEE, New York, NY USA, 821–826. https://doi.org/10.1109/MILCOM47813.2019.9020984

[54] Cankut Orakcal and David Starobinski. 2014. Jamming-resistant rate adaptation in Wi-Fi networks. *Performance Evaluation* 75-76 (5 2014), 50–68. https://doi.org/10.1016/j.peva.2014.02.002

[55] Prometheus Authors. 2014. Prometheus. https://prometheus.io/

[56] Martin L. Puterman. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley & Sons, Hoboken, New Jersey, USA.

[57] Partha Pratim Ray. 2022. A review on 6G for space-air-ground integrated network: Key enablers, open challenges, and future direction. *Journal of King Saud University - Computer and Information Sciences* 34 (10 2022), 6949–6976. Issue 9. https://doi.org/10.1016/j.jksuci.2021.08.014

[58] Ziheng Ren and Jiangfeng Xu. 2023. Research on the implementation of Kubernetes automatic scaling. In *Second International Conference on Electronic Information Engineering, Big Data, and Computer Technology (EIBDCT 2023)* (Xishuangbanna, China), Xiangjie Kong (Ed.). SPIE, Bellingham, WA, USA, 50. https://doi.org/10.1117/12.2674779

[59] Khaldoun Senjab, Sohail Abbas, Naveed Ahmed, and Atta ur Rehman Khan. 2023. A survey of Kubernetes scheduling algorithms. *Journal of Cloud Computing* 12 (6 2023), 87. Issue 1. https://doi.org/10.1186/s13677-023-00471-1

[60] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *2020 USENIX Annual Technical Conference (USENIX ATC 20).* USENIX Association, 205–218. https://www.usenix.org/conference/atc20/presentation/shahrad

[61] Theodore J Sheskin. 2016. *Markov Chains and Decision Processes for Engineers and Managers.* CBC Press, Boca Raton, FL, USA.

[62] Shachar Siboni, Asaf Shabtai, Nils O. Tippenhauer, Jemin Lee, and Yuval Elovici. 2016. Advanced Security Testbed Framework for Wearable IoT Devices. *ACM Transactions on Internet Technology* 16 (12 2016), 1–25. Issue 4. https://doi.org/10.1145/2981546

[63] Arunesh Sinha, Fei Fang, Bo An, Christopher Kiekintveld, and Milind Tambe. 2018. Stackelberg Security Games: Looking Beyond a Decade of Success. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence* (Stockholm, Sweden). International Joint Conferences on Artificial Intelligence Organization, 5494–5501. https://doi.org/10.24963/ijcai.2018/775

[64] Eilon Solan. 2003. Continuity of the Value of Competitive Markov Decision Processes. *Journal of Theoretical Probability* 16 (10 2003), 831–845. Issue 4. https://doi.org/10.1023/B:JOTP.0000011995.28536.ef

[65] Uma Tadakamalla and Daniel A. Menasce. 2022. Autonomic Resource Management for Fog Computing. *IEEE Transactions on Cloud Computing* 10 (10 2022), 2334–2350. Issue 4. https://doi.org/10.1109/TCC.2021.3064629

[66] Laszlo Toka, Gergely Dobreff, Balazs Fodor, and Balazs Sonkoly. 2021. Machine Learning-Based Scaling Management for Kubernetes Edge Clusters. *IEEE Transactions on Network and Service Management* 18 (3 2021), 958–972. Issue 1. https://doi.org/10.1109/TNSM.2021.3052837

[67] João Tomé. 2024. From Deals to DDoS: exploring Cyber Week 2024 Internet trends. https://blog.cloudflare.com/from-deals-to-ddos-exploring-cyber-week-2024-internet-trends/

[68] Thomas Tournaire, Hind Castel-Taleb, and Emmanuel Hyon. 2023. Efficient Computation of Optimal Thresholds in Cloud Auto-scaling Systems. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems* 8 (12 2023), 1–31. Issue 4. https://doi.org/10.1145/3603532

[69] Gianluca Turin, Andrea Borgarelli, Simone Donetti, Einar Broch Johnsen, Silvia Lizeth Tapia Tarifa, and Ferruccio Damiani. 2020. *A Formal Model of the Kubernetes Container Framework.* Springer, Cham, 558–577. https://doi.org/10.1007/978-3-030-61362-4_32

[70] Emmanouil Vasilomanolakis, Noorulla Sharief, and Max Muhlhauser. 2017. Defending against Probe-Response Attacks. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)* (Lisbon, Portugal). IEEE, New York, NY, USA, 1046–1051. https://doi.org/10.23919/INM.2017.7987436

[71] Joakim von Kistowski, Simon Eismann, Norbert Schmitt, Andre Bauer, Johannes Grohmann, and Samuel Kounev. 2018. TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)* (Milwaukee, WI, USA). IEEE, New York, NY, USA, 223–236. https://doi.org/10.1109/MASCOTS.2018.00030

[72] Xiaoqiong Xu, Jin Li, Hongfang Yu, Long Luo, Xuetao Wei, and Gang Sun. 2020. Towards Yo-Yo attack mitigation in cloud auto-scaling mechanism. *Digital Communications and Networks* 6 (8 2020), 369–376. Issue 3. https://doi.org/10.1016/j.dcan.2019.07.002

[73] Guangba Yu, Pengfei Chen, and Zibin Zheng. 2019. Microscaler: Automatic Scaling for Microservices with an Online Learning Approach. In *2019 IEEE International Conference on Web Services (ICWS)* (Milan, Italy). IEEE, New York, NY, USA, 68–75. https://doi.org/10.1109/ICWS.2019.00023

A DEFENDER STAGE MDP TRANSITION DEFINITIONS

Here, we define the transition probabilities for the Defender Stage SMDP from Section 4.2.1. Recall that for states $s \longrightarrow t \in \mathcal{S}$, the possible transition triggers are a single arrival or a single departure, yielding the following (Equation (3)):

$$\begin{cases} \lambda & \text{if } t = (m+d, \min(n+1, N)), \\ \min(n, m+d)\mu & \text{if } t = (m+d, \max(n-1, 0)). \end{cases}$$

Then given these transition rates, and the normalization factor defined in Equation (5):

$$\tilde{\Lambda} = \lambda + M\mu.$$

We have the ability to define a transition between arbitrary states $s \longrightarrow t \in \mathcal{S}$, given decision d. However, as part of the SMDP process, in addition to the probability of an arrival or departure occurring, we must also account for the probability that no transition occurs at the current time. That is $s = t$, and the system remains in the current state. With this in mind, we have the following general form for the transitions:

$$\mathcal{P}(t|s,d) = \begin{cases} \lambda/\tilde{\Lambda} & \text{if } t = (m+d, n+1), \\ \big(\min(n, m+d)\mu\big)/\tilde{\Lambda} & \text{if } t = (m+d, n-1), \\ 1 - \big(\lambda + \min(n, m+d)\mu\big)/\tilde{\Lambda} & \text{if } t = s, \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

If the queue is empty however, and $n = 0$, only arrivals are possible, which results in the following special case:

$$\mathcal{P}(t|s,d) = \begin{cases} \lambda/\tilde{\Lambda} & \text{if } t = (m+d, 1), \\ 1 - \lambda/\tilde{\Lambda} & \text{if } t = s, \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

Similarly, if the queue is full, and $n = N$, any arrivals are rejected and do not change the queue size. While the *cost function* is impacted by the attempted arrivals, the cluster configuration does not change. This results in the following:

$$\mathcal{P}(t|s,d) = \begin{cases} \big(\min(n, m+d)\mu\big)/\tilde{\Lambda} & \text{if } t = (m+d, N-1), \\ 1 - \big(\min(n, m+d)\mu\big)/\tilde{\Lambda} & \text{if } t = s, \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

B ATTACKER STAGE MDP TRANSITION DEFINITIONS

The attacker SMDP state transition probabilities for states $s \longrightarrow t \in \mathcal{S}^\pi$ are defined according to a similar process as the defender SMDP. The main difference is that, as per Equation (7), the rate of arrivals depends on whether the attacker is active or not:

$$\begin{cases} \lambda & \text{if } t = (m+\pi(s), \min(n+1, 0)), \text{and } a = 0; \\ (K+1)\lambda & \text{if } t = (m+\pi(s), \min(n+1, 0)), \text{and } a = 1; \\ \min(n, m+\pi(s))\mu & \text{if } t = (m+\pi(s), \max(n-1, 0)). \end{cases}$$

Thus, given the state transition rates, and the corresponding normalization rate (Equation (9)):

$$\tilde{\Lambda}_a = (K+1)\lambda + M\mu,$$

arbitrary state transition probabilities are defined as follows:

$$\mathcal{P}(t|s,a) = \begin{cases} \lambda/\tilde{\Lambda}_a & \text{if } t = (m+\pi(s), n+1) \text{ and } a = 0, \\ ((K+1)\lambda)/\tilde{\Lambda}_a & \text{if } t = (m+\pi(s), n+1) \text{ and } a = 1, \\ \big(\min(n, m+\pi(s))\mu\big)/\tilde{\Lambda}_a & \text{if } t = (m+\pi(s), n-1), \\ 1 - \Lambda_a(s,a)/\tilde{\Lambda}_a & \text{if } t = s, \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

As the attacker impacts arrival rate, two cases are required when considering arrivals, which is the key difference between the transition probabilities in Equations (14) and Equations (11). This in turn impacts the self transition rate - while this is implicitly the case due to defining self transitions in terms of the total state transition rate, we explicitly outline the impact by denoting each case separately in listing the probabilities for the special case of the empty buffer, and only arrivals are possible:

$$\mathcal{P}(t|s,a) = \begin{cases} \lambda/\tilde{\Lambda}_a & \text{if } t = (m+\pi(s), 1) \text{ and } a = 0, \\ ((K+1)\lambda)/\tilde{\Lambda}_a & \text{if } t = (m+\pi(s), 1) \text{ and } a = 1, \\ 1 - \lambda/\tilde{\Lambda}_a & \text{if } t = s, \text{ and } a = 0 \\ 1 - ((K+1)\lambda)/\tilde{\Lambda}_a & \text{if } t = s, \text{ and } a = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

However, when the buffer is full, it is not relevant whether the attacker is present or not, as arrivals are rejected and therefore the queue size does not change (the *reward function* is impacted as noted in Section 4.2.2, but this is a distinct consideration). As a result, the transition probabilities for the full buffer case is defined analogously to the case from Equation (13):

$$\mathcal{P}(t|s,a) = \begin{cases} \big(\min(n, m+\pi(s))\mu\big)/\tilde{\Lambda}_a & \text{if } t = (m+\pi(s), N-1), \\ 1 - \big(\min(n, m+\pi(s))\mu\big)/\tilde{\Lambda}_a & \text{if } t = s, \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

C SEMI MDP VALUE ITERATION ALGORITHM

Here we describe the discounted Value Iteration algorithms [56, p. 161] as applied to each SMDP. The *discount* refers to a parameter $\beta \in [0, 1)$ dictating the extent to which the algorithm favors greedy, short-term solutions over patient, long-term gains. β values closer to 0 correspond to the greedy version, β values closer to 1 correspond to the patient version.

For the Defender Stage SMDP, defined in terms of state space $\mathcal{S}$, decision space $\mathcal{D}$, and objective function C governing operating costs, the goal is to determine the policy π leading to the series of decisions $d \in \mathcal{D}$ at each state $s \in \mathcal{S}$ minimizing $C(s, d)$. The algorithm proceeds as follows:

(1) Let $n = 0$, specify $\epsilon > 0$, and let V^0 be an array of arbitrary initial values for each state.
(2) For each state $s \in \mathcal{S}$, compute $V^{n+1}(s)$:

$$V^{n+1}(s) = \min_{d \in \mathcal{D}} \left(C(s, d) + \sum_{t \in \mathcal{S}} \beta \mathcal{P}(t|s, d) V^n(t) \right).$$

(3) If the following holds:

$$||V^{n+1} - V^n|| < \epsilon(1 - \beta)/2\beta,$$

proceed to next step. Else, increment n by 1, and repeat step 2.
(4) The optimal value is expressed as V^{n+1}, and the corresponding policy π is defined at each state:

$$\pi(s) = \arg\min_{d \in \mathcal{D}} \left(C(s, d) + \sum_{t \in \mathcal{S}} \beta \mathcal{P}(t|s, d) V^{n+1}(t) \right).$$

For the Attacker Stage SMDP, defined in terms of $\mathcal{S}^\pi \subset \mathcal{S}$, decision space $\mathcal{A}$, and objective function $\mathcal{R}$ governing the net reward of attack, the algorithm has the goal to return policy π_a leading to the $a \in \mathcal{A}$ which maximizes $\mathcal{R}(s, a)$. The algorithm proceeds similarly to the Defender Stage case, except that steps (2) and (4) are modified accordingly:

(2) For each state $s \in \mathcal{S}^\pi$, compute $V^{n+1}(s)$:

$$V^{n+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}(s, a) + \sum_{t \in \mathcal{S}} \beta \mathcal{P}(t|s, a) V^n(t) \right).$$

(4) The optimal value is expressed as V^{n+1}, the corresponding policy π_a is defined at each state:

$$\pi_a(s) = \arg\max_{a \in \mathcal{A}} \left(\mathcal{R}(s, a) + \sum_{t \in \mathcal{S}} \beta \mathcal{P}(t|s, a) V^{n+1}(t) \right).$$

The validity of the discounted Value Iteration algorithm approach is justified by the following:

THEOREM C.1. *(Theorem 11.3.2.d.i in [56]) Suppose the following hypothesis holds. For all $s \in \mathcal{S}, d \in \mathcal{D}$, where F is the probability that the next decision occurs within δ time units of the present moment:*

(1) There exists $C^* > 0$ such that $|C(s, d)| \leq C^* < \infty$
(2) There exist $\delta, \epsilon > 0$ such that $F(\delta|s, d) \leq 1 - \epsilon$

Then there exists an optimal stationary deterministic policy for a discounted SMDP whenever $\mathcal{D}_s$ is finite for each $s \in \mathcal{S}$.

As both SMDPs have finite decision spaces, for each it suffices to show the hypothesis holds.

COROLLARY C.2. *The defender SMDP satisfies the conditions of Theorem C.1*

Proof. As the state space is finite, it can be established that $C(s,d)$ must be a bounded function. Given $C(s,d)$ as defined in Equation (6):

$$C(s,d) = \Big((m+d)C_S + \Lambda(s,d)\mathbb{1}_{d\neq 0}C_D + (n-\lambda(m+d)W)\mathbb{1}_{\frac{n}{(m+d)\lambda}>W}C_P + \lambda\mathbb{1}_{n=N}C_R\Big)\Big/\tilde{\Lambda}$$

whether C_D and C_R are active or not are binary decisions depending on a specific state-decision combination. Conversely, the cost of Service Unit operation C_S scales based on the number of SUs, while the delay penalty cost C_P also scales based on queue size subject to the SLA threshold. As $\tilde{\Lambda}$ is fixed, we conclude the following must hold:

$$C(s,d) \leq \Big(MC_S + \tilde{\Lambda}C_D + NC_P + \lambda C_R\Big)\Big/\tilde{\Lambda}$$

based on the maximum queue size N, Service Unit limit M, and potential action decisions and minimum SLA threshold $W=0$. The right hand side of the equation is thus our C^*, which must be finite, as the relevant costs and the normalization factor are themselves finite.

To satisfy the second condition, we note that $F(\delta|s,d)$ is the complementary probability of remaining in state s for δ time units. If $m+d<M$ or $n<M$, this can be defined directly in terms of the state transitions from Equations (11)- (13):

$$F(\delta|s,d) = 1-(1-\Lambda(s,d)/\tilde{\Lambda})^{\delta}$$

and $\epsilon = (1-\Lambda(s,d)/\tilde{\Lambda})^{\delta} > 0$. If however $m+d=M$ and $n\geq M$, $\Lambda(s,d)=\tilde{\Lambda}$ and this method fails. The state of the system in this scenario is that of an $M/M/c/K$ queue with M servers and $M\leq n\leq N$ requests in the system. Thus, as an alternative we leverage the steady state probabilities for such queues [5]. Let θ_i be the probability of i requests in the queue:

$$\theta_0 = \left[\sum_{k=1}^{M}\frac{\lambda^k}{\mu^k k!} + \frac{\lambda^M}{\mu^M M!}\sum_{M+1}^{N}\frac{\lambda^{k-M}}{\mu^{k-M}M^{k-M}}\right]^{-1} \tag{17}$$

$$\theta_i = \begin{cases}\frac{(\lambda/\mu)^i}{i!}\theta_0 & \text{for } i=1,2,\cdots,M\\ \frac{(\lambda/\mu)^i}{M^{i-M}M!}\theta_0 & \text{for } i=M+1,\cdots,N\end{cases} \tag{18}$$

Given Equations (17) and (18), let $F(\delta|s,d) = 1-(\theta_n)^{\delta}$, then $\epsilon=(\theta_n)^{\delta}>0$ satisfies the condition. □

Corollary C.3. *The attacker SMDP satisfies the conditions of Theorem C.1*

Proof. For the first condition, replacing $C(s,d)$ with $\mathcal{R}(s,a)$ as defined in Equation (10):

$$\begin{aligned}\mathcal{R}(s,a) =& \Big((m+\pi(s))C_S + \Lambda_a(s,a)\mathbb{1}_{\pi(s)\neq 0}C_D + (1+K\mathbb{1}_{a=1})\lambda\mathbb{1}_{n=N}C_R\\ &+ \big(n-(m+\pi(s))(K\mathbb{1}_{a=1}+1)\lambda W\big)\mathbb{1}_{\frac{n}{(m+\pi(s))(K\mathbb{1}_{a=1}+1)\lambda}>W}C_P - K\mathbb{1}_{a=1}C_A\Big)\Big/\tilde{\Lambda}_a.\end{aligned}$$

C_A is the only negative cost, but only applies if $a=1$. As a result, we can define a bound R^* which is very similar to the C^* defined for Corollary C.2:

$$R^* = (MC_S + \tilde{\Lambda}_a C_D + NC_P + (K+1)\lambda C_R)/\tilde{\Lambda}_a$$

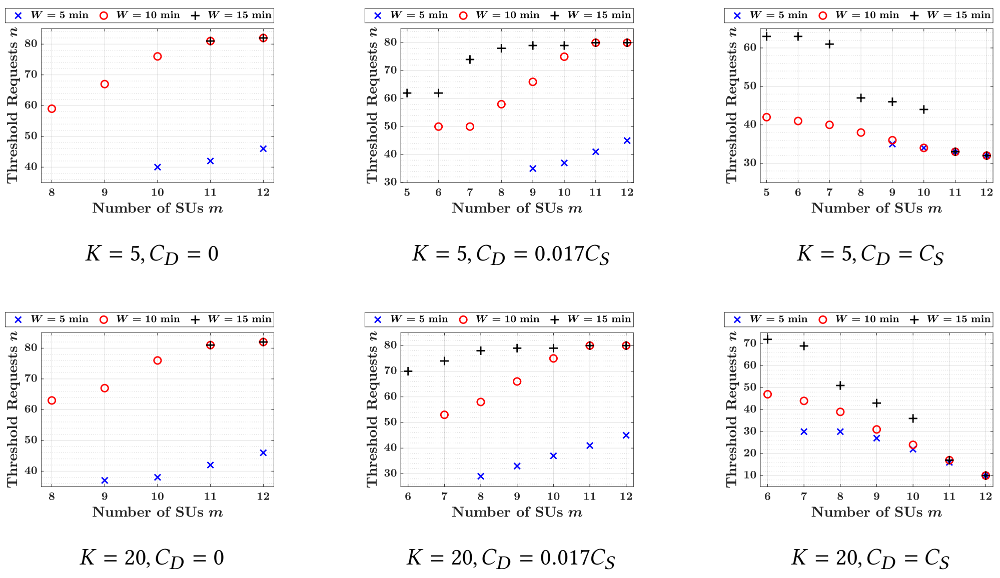
and $|\mathcal{R}(s,d_a)| \leq R^* < \infty$ must hold by the same logic which defined C^*, as the attacker reward is ultimately defined in terms of the attacker cost in addition to the costs of generating extra traffic; the sole distinction being between in which direction the quantity should be optimized.

For the second condition, while a new transition condition exists, the types of transitions have not changed, only the rate of arrivals if a particular action has been taken. As a result, we leverage similar logic as under Corollary C.2. Given $\tilde{\Lambda}_a = (K+1)\lambda + M\mu$, then when $m_{\pi(s)} < M$ or $n<M$ or $a=0$, we define $F(\delta|s,a)$ in terms of the self transition, where $\epsilon = (1-\Lambda(s,a)/\tilde{\Lambda}_a)^{\delta} > 0$. Otherwise,

δ, ϵ and F are defined in terms of the $M/M/c/K$ queue stationary probabilities in Equations (17) and (18) as before. $\square$

D SUPPLEMENTAL RESULTS

Here we present the supplementary tables and plots for the evaluations in Section 3 for the numerical evaluation examples, and Section 5 for the experimental evaluations. Figures 7 contains the plots consisting of the threshold number of requests at each level to launch an attack under the Model A, $M = 12, N = 100$ configuration for attack powers $K \in \{5, 20\}$. Tables 9 contains the minimum thresholds (m, n), for the minimum SU level m, for the attack across each model and attack power for the $W \in \{5, 10\}$ min SLA thresholds. Figures 8 and 9 contain plots of the Service Unit utilization under attack conditions under the Bursty and On-Off customer traffic patterns in our experimental evaluations; under the latter, the plot demonstrates that SU utilization remains under 5 SUs and thus, the trigger is never reached to initiate the attack.



$K = 5, C_D = 0$ $K = 5, C_D = 0.017C_S$ $K = 5, C_D = C_S$

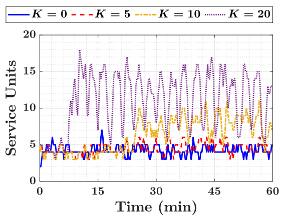
$K = 20, C_D = 0$ $K = 20, C_D = 0.017C_S$ $K = 20, C_D = C_S$

Fig. 7. Attack thresholds for the Model A, $M = 12, N = 100$ configuration and attack power $K \in \{5, 20\}$ given the parameters in Table 3.

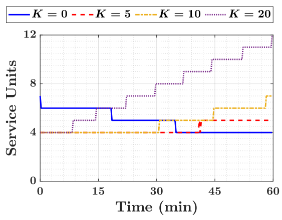
Received January 2025; revised April 2025; accepted April 2025

Model Configuration		$W = 5$ min			$W = 10$ min		
		$C_D = 0$	$C_D = 0.017C_S$	$C_D = C_S$	$C_D = 0$	$C_D = 0.017C_S$	$C_D = C_S$
$K = 5$	Model A $M = 12, N = 100$	$(10, 40)$	$(9, 35)$	$(9, 35)$	$(8, 59)$	$(6, 50)$	$(5, 42)$
	Model A $M = 16, N = 100$	$(13, 54)$	$(12, 46)$	$(11, 46)$	$(15, 88)$	$(8, 66)$	$(8, 66)$
	Model A $M = 32, N = 200$	$(32, 162)$	$(32, 162)$	$(32, 162)$	$(32, 195)$	$(23, 189)$	$(22, 180)$
	Model A $M = 64, N = 400$	$(64, 385)$	$(64, 385)$	$(64, 385)$	$(63, 397)$	$(62, 397)$	$(54, 398)$
	Model B	$(6, 24)$	$(6, 24)$	$(6, 24)$	$(4, 32)$	$(4, 26)$	$(3, 26)$
	Model C	$(3, 5)$	$(2, 5)$	$(1, 4)$	$(2, 9)$	$(1, 6)$	$(1, 5)$
$K = 10$	Model A $M = 12, N = 100$	$(9, 37)$	$(8, 29)$	$(7, 30)$	$(8, 59)$	$(6, 48)$	$(5, 41)$
	Model A $M = 16, N = 100$	$(13, 50)$	$(10, 40)$	$(10, 39)$	$(15, 88)$	$(8, 66)$	$(8, 66)$
	Model A $M = 32, N = 200$	$(32, 136)$	$(32, 136)$	$(32, 136)$	$(32, 195)$	$(22, 183)$	$(22, 180)$
	Model A $M = 64, N = 400$	$(64, 363)$	$(64, 363)$	$(64, 363)$	$(63, 397)$	$(62, 397)$	$(54, 398)$
	Model B	$(5, 17)$	$(4, 13)$	$(3, 13)$	$(4, 26)$	$(3, 18)$	$(3, 17)$
	Model C	$(3, 5)$	*	*	$(1, 2)$	*	*
$K = 20$	Model A $M = 12, N = 100$	$(9, 37)$	$(8, 29)$	$(7, 30)$	$(8, 63)$	$(7, 53)$	$(6, 47)$
	Model A $M = 16, N = 100$	$(13, 50)$	$(10, 41)$	$(10, 41)$	$(15, 88)$	$(9, 72)$	$(9, 73)$
	Model A $M = 32, N = 200$	$(31, 125)$	$(29, 121)$	$(29, 123)$	$(32, 195)$	$(22, 183)$	$(20, 184)$
	Model A $M = 64, N = 400$	$(64, 332)$	$(64, 332)$	$(64, 332)$	$(63, 397)$	$(62, 397)$	$(47, 396)$
	Model B	$(4, 15)$	$(3, 10)$	$(3, 11)$	$(4, 26)$	$(3, 17)$	$(2, 17)$
	Model C	$(3, 5)$	*	*	*	*	*

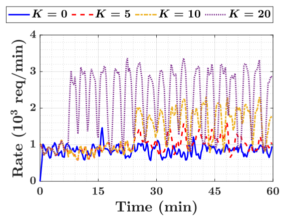
Table 9. Threshold state (m, n) corresponding to the minimum SUs at which the attacker will initiate the attack decision under each cluster configuration, and minimum cost C_D combination for the $W \in \{5, 10\}$ min SLA threshold scenarios. The asterisks indicate scenarios in which it is universally beneficial to initiate an attack.



Per-minute SU utilization



Mean per-hour SU utilization



Mean Request Rate

Fig. 8. The Service Rate utilization per-minute and mean per-hour, and the corresponding request rates for each attack strength compared to the baseline utilization ($K = 0$) under Bursty customer traffic.

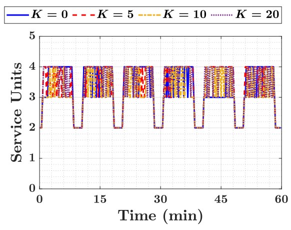


Fig. 9. The Service Rate utilization per-minute under On-Off customer traffic, demonstrating that SU utilization never reaches a level where the attacker will initiate the attack.