Bankrupting DoS Attackers

Trisha Chakraborty, Abir Islam, Valerie King, Daniel Rayborn, Jared Saia, Maxwell Young

[a] *Amazon Web Services, Minneapolis, , MN, USA*
[b] *Meta Research, Menlo Park 94025, CA, USA*
[c] *Department of Computer Science, University of Victoria, Victoria, V8P 5C2, BC, Canada*
[d] *Department of Computer Science and Engineering, Mississippi State University, Mississippi State, 39762, MS, USA*
[e] *Department of Computer Science, University of New Mexico, Albuquerque, 87131, NM, USA*
[f] *Department of Computer Science and Engineering, Mississippi State University, Mississippi State, 39762, MS, USA*

Abstract

Can we make a denial-of-service attacker pay more than the server and honest clients? Consider a model where a server sees a stream of jobs sent by either honest clients or an adversary. The server sets a price for servicing each job with the aid of an *estimator*, which provides approximate statistical information about the distribution of previously occurring good jobs.

We describe and analyze pricing algorithms for the server under different models of synchrony, with total cost parameterized by the accuracy of the estimator. Given a reasonable accurate estimator, the attacker's cost grows asymptotically faster than our algorithm's cost. Additionally, we prove a lower bound, showing that our pricing algorithm yields asymptotically tight results when the estimator is accurate within constant factors.

Keywords: Denial-of-service attacks, Resource burning, Distributed algorithms.

1. Introduction

Since their first documented occurrence in 1974 [61], denial-of-service(DoS) attacks have evolved into a multi-billion dollar security dilemma [105]. The DoS problem has always been economic: success or failure is decided by who can marshal resources most effectively. In this paper, we provide a novel theoretical formulation and analysis of the DoS problem. We hope this will be a first step towards: (1) leveraging tools from theoretical computer science to solve a critical and persistent security problem; and (2) spurring new theoretical techniques and models; for example, in the same way that the theoretical focus on the paging problem eventually helped create the area of online algorithms.

It has long been known that fees can be used to curb DoS attacks [44, 62, 113, 95]. This fee may be in the form of computation: a client solves a computational puzzle or Captcha per request [64, 43, 81]; or bandwidth: a client sends multiple messages to the server per request [114, 113, 125]. Fees may be set to 0 in the absence of attack, so that in peaceful times, clients pay nothing.

In this paper, we design algorithms for setting the fee (the "price") in an online manner, assisted by an "estimator", which provides information about the number of previously occurring good jobs in any past time interval. Our goal is to set prices so as to minimize the total algorithm's cost—that is, the cost to the server and honest clients—relative to the attacker's cost. To achieve this, we define an accuracy measure of the estimator, and determine how the algorithm's cost increases with this measure.

1.1. Our Model and Problem

1.1.1. Server, Clients, and Communication

The ***server*** receives a stream of jobs. Each job is either ***good***: sent by a client—to avoid redundancy, henceforth "clients" rather than "honest clients"—or ***bad***: sent by the adversary (described below). The clients start running our client algorithm when they arrive; the server runs our server algorithm at the beginning of time (see Section 3).

Communication occurs via message-passing. The following happens in parallel for the server and clients.

- **Server:** The server uses the estimator to compute the current price. Jobs are serviced based on whether the attached fee meets this price.

When the server receives a job request with an insufficient fee, it sends the current price back to the sender.

- **Clients:** Each client may send a job to the server with a fee.

1.1.2. Adversary

The adversary sets the times at which all good jobs arrive, and also sets all outputs of the estimator, subject to the estimation gap γ and the additivity constraint. Also, the adversary knows everything about our algorithm. Finally, at any time, the adversary can send messages to the server for (bad) job requests with fees attached.

1.1.3. Costs

Our algorithmic costs are two-fold: (1) fees paid by clients; and (2) ***service costs*** incurred by the server. We assume a normalized service cost of 1 for servicing any job, whether good or bad. The cost to the adversary is the total cost of all fees paid to the server.

1.1.4. Estimator

The server has an estimator whose output may be used to set prices. The ***estimator*** estimates the number of good jobs generated in any contiguous time interval in the past. For any such interval, $\boldsymbol{\mathcal{I}}$, let $\boldsymbol{g(\mathcal{I})}$ be the number of good jobs in $\mathcal{I}$ and $\boldsymbol{\hat{g}(\mathcal{I})}$ be the value returned by the estimator for interval $\mathcal{I}$. The estimator $\hat{g}$ is additive: for any two non-overlapping intervals, $\mathcal{I}_1$ and $\mathcal{I}_2$:

$$\hat{g}(\mathcal{I}_1 \cup \mathcal{I}_2) = \hat{g}(\mathcal{I}_1) + \hat{g}(\mathcal{I}_2).$$

We provide examples of estimators that satisfy the additive property in Section 5. For a given input, the ***estimation gap***, $\boldsymbol{\gamma}$, is the smallest number greater than or equal to 1, such that the following holds for all intervals, $\mathcal{I}$ in the input:

$$g(\mathcal{I})/\gamma - \gamma \leq \hat{g}(\mathcal{I}) \leq \gamma g(\mathcal{I}) + \gamma.$$

The estimator is allowed to be probabilistic, so long as it satisfies additivity. The estimation gap value of γ achieved for a particular input stream of jobs will determine the resource cost of the algorithm, as described in main upper bounds provided in Section 1.2.

1.1.5. Time and Communication Latency.

We assume that time is continuous; good jobs arrive at times set by the adversary; and that the server has access to a global clock. For communication latency, we have two problem variants:

- **Zero-latency:** All messages are sent instantaneously. So, the clients always know the exact current price; and the fees attached to jobs sent to the server always arrive before any change in the current price, and thus are always serviced.

- **Asynchronous:** Communication is *asynchronous* [85]: there is some unknown maximum latency, $\boldsymbol{\Delta}$ seconds, for any message to be received after it is sent. This latency can also capture the maximum time it took any client to attach a fee. Also, there is some unknown maximum number of good jobs, $\boldsymbol{M}$, that can be generated over Δ seconds. Both Δ and M are known to the adversary, but not to the algorithm. Also, the adversary controls the timing of all messages subject to the above constraints.

1.1.6. Our Goal

Our primary goal is to minimize the sum of the costs of the server and clients as a function of the adversary's cost. Secondary goals are to minimize the number of messages sent and the maximum number of messages that must be exchanged before a client has its job serviced.

1.2. Our Results

For a given input, let $\boldsymbol{g}$ be the number of good jobs, $\boldsymbol{B}$ be the total spent by the adversary, and γ be the estimation gap for the input. Our primary goal is to minimize the algorithmic cost—i.e. fees plus service costs—as a function of g, B, and γ

Our first result is for the algorithm, LINEAR, which assumes zero-latency.

Theorem 1. LINEAR *has total cost*

$$O\left(\gamma^{5/2}\sqrt{B\,(g+1)}+\gamma^3(g+1)\right).$$

Moreover, the server sends $O(g)$ messages to clients, and clients send a total of $O(g)$ messages to the server. Also, each good job is serviced after at most 3 *messages are exchanged.*

To build intuition for this result, consider the case where $\gamma = O(1)$. Then, the theorem states that the cost to LINEAR is $O(\sqrt{B(g+1)} + (g+1))$. Importantly, the cost to the algorithm grows asymptotically slower than B whenever $g = o(B)$. We note that the additive $O(g+1)$ term results from service costs due to good jobs, which must hold for any algorithm. Our result is actually better than stated in Theorem 1, when γ grows large; in that case the total cost is always bounded by $g^2 + B$, as will become clear in Section 3.

Finally, the number of messages sent from server to clients and from clients to server, and also the maximum number of messages exchanged to obtain service are all asymptotically optimal. To see this note that at least one message must be sent for each good job serviced.

What if there is latency? For our asynchronous variant, we design and analyze an algorithm LINEAR-POWER. Our main theorem for this case follows. The $\tilde{O}$ notation for the algorithmic cost hides terms that are logarithmic in B and γ.

Theorem 2. LINEAR-POWER *has total cost:*

$$\tilde{O}\left(\gamma^3\left(\sqrt{B+1}\min\left(g+1, M\sqrt{g+1}, M\sqrt{B+1}\right) + (g+1)\right)\right).$$

Moreover, the server sends $O(g\log(B+\gamma))$ messages to clients, and clients sends a total of $O(g\log(B+\gamma))$ messages to the server. Also, each good job is serviced after at most $4\log(\gamma(B+1))+3$ messages are sent.

To gain intuition for this result, assume that γ and M are both $O(1)$. Then the cost to LINEAR-POWER is $\tilde{O}(\sqrt{B+1}\,\min(\sqrt{g+1}, \sqrt{B+1}) + (g+1))$. This equals $\tilde{O}(\sqrt{B(g+1)} + (g+1))$, since when $B = O(g)$, the expression is $\tilde{O}(g+1)$; and otherwise the expression is $\tilde{O}(\sqrt{B(g+1)})$. Thus, when γ and M are constants, the cost of LINEAR-POWER matches that of LINEAR, up to logarithmic terms. For communication, the number of messages sent from the server to clients, clients to the server, and the number of messages exchanged until a job is serviced all increase by a logarithmic factor in B, when compared to LINEAR.

To complement these upper bounds, in Section 8, we also prove the following lower bound. This lower bound holds for our zero-latency variant, and so directly also holds for the harder, asynchronous variant. Let $\boldsymbol{n}$ be the total number of jobs.

Theorem 3. *Let γ be any positive integer; g_0 be any positive multiple of γ; and n any multiple of γg_0. Next, fix any randomized algorithm. Then, there is an input with n jobs, g of which are good for $g \in \{\gamma g_0, g_0/\gamma\}$; an estimator with gap γ on that input; and the randomized algorithm on that input has expected cost:*

$$\Omega\left(\sqrt{\gamma B(g+1)} + \gamma(g+1)\right).$$

When $\gamma = O(1)$, this lower bound asymptotically matches the cost for LINEAR, and is within logarithmic factors of the cost for LINEAR-POWER. Additionally, the lower bound allows for a fair amount of independence between the settings of γ, g and n for which the lower bound can be proven. For example, it holds both when g is small compared to n and also when g is large. Finally, note that the lower bound holds for randomized algorithms, even though both LINEAR and LINEAR-POWER are deterministic. Thus, randomization does not offer significant improvements for our problem.

Finally, in Section 9, we provide preliminary simulation results for LINEAR and LINEAR-POWER. These results are illustrative of the type of results predicted by theory, and they suggest that the hidden constants in the asymptotic notation may be small, and that the scaling behavior for our algorithms' costs matches that predicted by theory.

1.3. Model Justification

(Lack of) Knowledge of the Servers and Clients. Other than the outputs of the estimator, the server and clients have no prior knowledge of the number of good jobs, the amount spent by the adversary, or the estimation gap γ. Recall that, in the latency model, Δ and M are also unknown.

Estimator. Our estimator generalizes many DoS detection tools, summarized below; see also [66, 78] for comprehensive surveys. DoS detection tools can be characterized along 3 main axes. First, input attributes used include: network traffic patterns [41, 127, 116, 93, 107, 18, 30, 122, 104, 29, 77, 37]; reliability of access routers [55]; request payloads [92, 69, 125]; geographical information [10]; hop count travelled by request messages [33, 115]; and social media posts [26]. Second, detection algorithms used include: Bayesian learning [109, 65, 55]; k-nearest neighbors [93, 11, 79, 10, 125]; neural networks and deep learning [41, 127, 26, 30, 122]; and statistical methods [92, 104, 29, 77, 37, 116]. Finally, outputs of the detection algorithms include:

classification of each packet as good or bad [41, 127, 109, 122, 104, 29, 125]; or general detection of the occurrence of an attack or the severity of attack [65, 116, 10, 93, 107, 18, 26, 30, 122, 104, 29, 77, 37].

The estimator generalizes three detection techniques. First, for detection tools that **classify jobs**: the estimator can return the number of jobs classified as good in the input interval, perhaps adjusted based on the classifier error rates if these are known. Second, for detection tools that **detect attacks**: when there is no attack detected in the input interval, the estimator can return the number of jobs in the interval; when there is an attack detected, the estimator can return the expected number of good jobs based on past data before the attack. Finally, when good jobs arrive via a **Poisson, Weibull, Gamma, or similar stochastic process**: we describe an estimator, in Section 5, with estimation gap that is $O(\log g)$, with probability at least $1 - 1/g^k$ for any positive k, where g is the number of good jobs over the system lifetime.

Prices. Many DoS defenses incorporate prices for service. One such tool is **bandwidth-based pricing** [78, 113, 114, 125, 84]. Every message from a customer incurs a resource cost to the customer; a bandwidth price for a job is set by requiring the customer to send a certain number of messages per job serviced. In the simplest case, the server services each request with some tunable probability. This sets an *expected* bandwidth cost for each job serviced. When this probability changes, it is shared with the customers. Speak-up [113, 114] (402 cites, according to Google Scholar) and DOW [125] (112 cites) are popular DoS defenses that primarily use this approach. When these systems detect an attack, they notify customers that the service probability has decreased. If the attack continues, good and bad customers pay similar bandwidth costs, and also have similar rates of service; thus the total algorithmic cost is $\Omega(B)$, where B is the amount spent by the attacker.

Rate-limiting is a bandwidth-based defense technique where packets can be dropped with some tunable probability, but this probability is not necessarily shared with the customers [82, 131, 69, 68]; OpenFlow [91, 18, 128, 19, 124] (> 1800 cites) is a popular such rate-limiting system, even though no theoretical guarantees have been established.

Puzzle-based pricing defenses require customers to solve computational or proof-of-work (PoW) puzzles in order to receive service [44, 62, 7, 28, 67, 72, 100, 40, 97]. Popular DoS defense systems using PoW include: Poseidon [132] (207 cites); KaPow [48, 63] (37 cites); Phalanx [40] (120 cites); Portcullis

[97] (288 cites); OverDOSe [100] (52 cites); and Puzzle Auctions [117] (304 cites). Other puzzle-based defenses use *Captchas* [110, 111]: human-solvable puzzles. Such systems including: Kill-Bots [64] (572 cites) and Enhanced DDoS-MS [4] (13 cites). In all of these puzzle-based systems, customers pay an amount during an attack that matches the attacker, and thus receive a fair share of job service. However, to the best of our knowledge, none of these defense systems have provable total cost of $o(B)$.

Asymptotic Advantage. Many DoS defenses **filter** out packets that are classified as bad [130]. But, even if filtering is 99.9% effective—and reported classification accuracy now hovers around 95% [41, 92, 127]—filtering alone still requires the defenders to spend $\Omega(B)$. Thus, standalone filtering will never give the defender an asymptotic advantage, although filtering can be combined with our results to reduce costs by constant factors.

Should DoS defenders care about an asymptotic advantage? Resource costs related to DoS attacks are increasingly liquid: attackers can rent botfarms on the black market [101]; and defenders can provision additional resources from the cloud to service requests [126, 133, 9]. So, if a defender has access to a DoS defense algorithm with an asymptotic cost advantage, it seems feasible they should be able to outlast an attacker with a similar monetary budget. See also Section 2 for game-theoretic implications.

2. Related Work

In addition to the discussion given previously in Section 1.1, we summarize additional related results here.

Game Theory. Many results analyze security problems in a game theoretic setting. These include two-player security games between an attacker and defender, generally motivated by the desire to protect infrastructure (airports, seaports, flights, networks) [103, 8, 112] or natural resources (wildlife, crops) [47, 46]. Game theoretic security problems may include multiple players, and may also have a mix of Byzantine, altruistic and rational (BAR) players [35, 34].

Specific to the challenge of mitigating DoS attacks, there are several results that apply game theory (for examples, [121, 95, 102, 45, 39, 60]). In contrast to these results, our approach does not assume the adversary, server or clients are rational. Our main results show how to minimize the server and clients costs *as a function of the adversarial cost*. This is different than

optimizing most typical utility functions for the defender. However, there are two game theoretic connections. First, when there exists an algorithm for the defender, as in this paper, where the defender pays some function f of the adversarial cost B, and $f(B) = f(0) + o(B)$, it is possible to solve a natural zero-sum, two-player attacker-defender game. The solution of this game is for the attacker to set $B = 0$ — essentially to not attack —-in which case the defender pays a cost of $f(0)$, which in our case is $O(\gamma^3 g)$ (see Section 2.6 of [57] for details). Second, our paper suggests, for future work, the game-theoretic problem of refining our current model and algorithms in order to build a mechanism for clients and a server that are all selfish-but-rational (Section 10).

General Resource Burning. Starting with the seminal 1992 paper by Dwork and Naor [44], there has been significant academic research, spanning multiple decades, on using resource burning (RB) —the verifiable expenditure of resources —to address many security problems; see surveys [3, 57]. Security algorithms that use resource burning arise in the domains of wireless networks [54], peer-to-peer systems [76, 15], blockchains [80], and e-commerce [57]. Our algorithm is purposely agnostic about the specific resource burned, which can include computational power [118], bandwidth [114], computer memory [1], or human effort [110, 96]. In the DoS setting, the functionality of puzzle generation and solution verification can be hardened against attack by outsourcing to a dedicated service [119]; leveraging the Domain Name System [75]; or using routers [106].

Resource-Competitive Algorithms. There is a growing body of research on security algorithms whose costs are parameterized by the adversary's cost. Such results are referred to as *resource-competitive* [14]. Specific results in this area address: communication on a broadcast channel [53, 51, 70, 27], contention resolution [13, 12], interactive communication [38, 2], hash tables [25], bridge assignment in Tor [129], and the Sybil attack [58, 59, 56]. Our result is the first in this area to use an estimator and incorporate the estimation gap into the algorithmic cost.

Algorithms with Predictions. Algorithms with predictions is a new research area which seeks to use predictions to achieve better algorithmic performance. In general, the goal is to design an algorithm that performs well when prediction accuracy is high, and where performance drops off gently with decreased prediction accuracy. Critically, the algorithm has no a priori knowledge of the accuracy of the predictor on the input.

The use of predictions offers a promising approach for improving algorithmic performance. This approach has found success in the context of contention resolution [52]; bloom filters [87]; online problems such as job scheduling [98, 74, 99]; and many others. See surveys by Mitzenmacher and Vassilvitskii [89, 90]. Our estimation gap for the input, γ is related to a accuracy metric for a predictor of job lengths in [99].

Our estimator differs from predictions in that it never returns results predicting the future, it only estimates the number of good jobs in past intervals.

3. Our Algorithms

In Section 3.1, we formally define the algorithm LINEAR; and in Section 3.2, we define the algorithm LINEAR-POWER.

3.1. Algorithm LINEAR

LINEAR addresses the zero-latency variant by partitioning job processing into ***iterations*** that terminate when $\hat{g}(\mathcal{I}) \geq 1$, where $\mathcal{I}$ is the time interval containing the current iteration. For the s-th job in an iteration, the server sets the price $\boldsymbol{PRICE} = s$. Under zero latency, clients face a binary choice: submit the job with a fee of PRICE or drop it. The server ignores any solutions below this threshold and breaks arrival ties arbitrarily (Figure 1).

3.2. Algorithm LINEAR-POWER

LINEAR-POWER addresses the asynchronous variant (Figure 2) using an iteration-based pricing scheme.

Server. Iterations terminate when $\hat{g}(\mathcal{I}) \geq 1$. For the s-th job in an iteration, the server sets PRICE $= 2^{\lfloor \log s+1 \rfloor}$. A job is serviced only if its attached fee is at least PRICE; otherwise, the server rejects the job and returns the current PRICE value to the client.

Client. Each client maintains a local fee x, initialized to 1. If a job is rejected with a threshold value $T > x$, the client updates $x \leftarrow T$ and either resubmits the job with the new fee or drops it.

LINEAR

Server:
Repeat forever:

1. If $\hat{g}(\mathcal{I}) \geq 1$, where $\mathcal{I} = (t, t']$, t is the start time of the current iteration, and t' is the current time, then begin a new iteration.
2. PRICE $\leftarrow s + 1$, where s is the number of jobs serviced so far in this iteration. If PRICE has changed, send the new value to all clients.
3. If the next job has a attached fee at least equal to PRICE then service it, otherwise do not service and reply with PRICE.

Client: $x \leftarrow 1$.
Until the job is serviced, do:

1. Send the server a job request with fee x.
2. $x \leftarrow$ most-recent price received from the server.

Figure 1: Pseudocode for LINEAR.

4. Technical Overview

Our full proofs are provided in Sections 6, 7, and 8. Here, we give intuition for our technical results.

Zero-Latency. Our analysis of LINEAR (Section 3.1) involves two key steps: (1) establishing upper and lower bounds on the number of iterations as a function of γ and g (Lemmas 1 and 3); and (2) deriving an upper bound on the number of good jobs in any iteration (Lemma 2).

Next, using (1) and (2), we prove an upper bound on the cost of LINEAR. To build intuition, we consider a general class of current price setting algorithms where the charge to the s-th job in the iteration is s^α, for any constant $\alpha \geq 0$. We prove that $\alpha = 1$ gives the best algorithmic cost as a function of B, g and γ.

Our upper bound consists of four exchange arguments [71], which will be over the ordering of good and bad job requests. Our first two exchange

LINEAR-POWER

Server:
Repeat forever:

1. If $\hat{g}(\mathcal{I}) \geq 1$, where $\mathcal{I} = (t, t']$, t is the start time of the current iteration, and t' is the current time, then begin a new iteration.
2. PRICE $\leftarrow 2^{\lfloor \lg(s+1) \rfloor}$, where s is the number of jobs serviced in this iteration.
3. If the next job has a fee at least equal to PRICE, then service it. Otherwise, do not service the job; instead send a message with the value PRICE to the client that sent the job.

Client: $x \leftarrow 1$.
Until the job is serviced, do:

1. Send the server a fee of x with the job.
2. $x \leftarrow$ maximum price ever received from the server.

Figure 2: Pseudocode for LINEAR-POWER.

arguments lower bound B and the next two upper bound the algorithm cost, $\boldsymbol{A}$. We begin by lower bounding B. To do so, our first exchange argument shows that B is minimized when, in each iteration, the bad jobs occur before the good. Our second exchange argument shows that, subject to this constraint, B is minimized when the bad jobs are distributed as evenly as possible across the iterations. From these two exchange arguments, a lower bound on B follows via an integral lower bound and basic algebra (see Lemma 4).

We next upper bound A. To do so, we make two observations: (1) within an iteration, A is maximized by putting all good jobs at the end; and (2) without loss of generality, we can assume that the iterations are sorted in decreasing order by the number of bad jobs they contain. Then, we use a third (simple) exchange argument to show that A is maximized when good jobs are "packed left": packed as much as possible in the earlier iterations, which have the higher number of bad jobs.

Finally, we use the fourth (harder) exchange argument to show that A is maximized when *bad* jobs are "packed left" when $\alpha \geq 1$, and are evenly spread when $\alpha < 1$. To show this, for an iteration i, we let $f(b_i)$ be the cost of the i-th iteration when there are b_i bad jobs at the start. Then, let $\delta(b_i) = f(b_i + 1) - f(b_i)$ be the change in the cost of iteration i when we add 1 bad job. Then the change in cost when there is an exchange of one bad job from iteration k to iteration i for $k > i$ will be $\delta(b_i) - \delta(b_k - 1)$.

How do we show that $\delta(b_i) - \delta(b_k - 1)) \geq 0$? If we relax b_i so it can take on values in the real numbers, then $f(b_i)$ becomes a continuous, differentiable function. Thus, we can use the mean value theorem (MVT). In particular, the MVT shows that $\delta(b_i)$ equals $f'(x)$ at some value of $x \in [b_i, b_{i+1}]$. Then, upper and lower bounding the δ values reduces to bounding f' in the appropriate range. This is a simpler problem because f' is a monotonically non-decreasing (non-increasing) function when $\alpha \geq 1$ ($\alpha < 1$, respectively). Once we have this exchange argument in hand, we can calculate the algorithmic costs (see the proofs of Lemma 5 and Theorem 1 in Section 6 for details).

Asynchronous. If we naively use LINEAR for the asynchronous model, costs can be high. For simplicity, assume that $\gamma = \Theta(1)$ and $g = \Theta(M\sqrt{B})$. Then, the adversary can insert $\sqrt{B}$ bad jobs initially, causing the current price in LINEAR to increase $\sqrt{B}$ times. During this time $\Theta\left(\min\left(g, M\sqrt{B}\right)\right)$ good jobs join the system, all of them are returned without service repeatedly, as the current price increases, until the current price reaches its maximum value of $\Theta(\sqrt{B})$. Next, all $\Theta\left(\min(g, M\sqrt{B})\right)$ of these good jobs wind up paying a fee equal to $\Theta(\sqrt{B})$. Adding in the cost to service jobs and the cost for any additional good jobs that enter after the bad, we have a total cost for the LINEAR of $\Theta\left(\min\left(g+1, M\sqrt{B+1}\right)\sqrt{B+1} + g\right)$.

How can we improve this? One key idea is to reduce the number of times that the server increases the current price. Our second algorithm, LINEAR-POWER does just that: (1) the server sets the current price to the largest power of 2 less than the total number of jobs in the current iteration; and (2) clients double their fee whenever their job request is returned.

To compare LINEAR-POWER with LINEAR, suppose $\gamma = O(1)$. Then, it is not hard to see that in any iteration i where the adversary spends B_i, the maximum current price is $O(\sqrt{B_i})$ (see Lemma 8). Thus, we can argue that any job is returned $O(\log B)$ times, and so pays a total of $O(\sqrt{B})$.

But the above argument gives a poor bound on fees paid by the clients,

since it multiplies all such fees from the LINEAR analysis by a $\sqrt{B}$ factor. To do better, we need a tighter analysis for fees. But how do we proceed when a single job can be refused service over many iterations because it keeps learning the current price too late?

The first key idea is to partition time into epochs, for the purpose of analysis only. An ***epoch*** is a period of time ending with 2Δ seconds during which the current price never increases. Then, several facts follow. First, every good job is serviced within 2 epochs. Second, we can bound the total fees that all good jobs pay during epoch i to be essentially $O(M(\sqrt{B_{i-1}} + \sqrt{B_i}))$, where B_j is the amount spent by the adversary in epoch j (see Lemma 11). Third, an epoch has overpaying jobs only if it or its preceding epoch spans the beginning of an iteration (see Lemma 10). This last fact allows us to bound the total number of epochs that add to the overpayment amount using our previous bounds on the number of iterations. Putting all these facts together, we obtain a total cost for LINEAR-POWER that is only $\tilde{O}\big(\min\big(g+1, M\sqrt{g+1}, M\sqrt{B+1}\big)\sqrt{B+1} + g\big)$ (see Section 7) versus the cost of LINEAR computed above of $\Theta(\min(g+1, M\sqrt{B+1})\sqrt{B+1} + g)$. Ignoring logarithmic factors, the cost of LINEAR-POWER may be less than LINEAR by a factor of $\sqrt{g+1}/M$. As an example, when $M = O(1)$ and $B = \Theta(g)$, LINEAR-POWER has cost $\tilde{O}(g)$, and LINEAR has cost $\Theta(g^{3/2})$.

Lower Bound. We prove a lower bound for the zero-latency case (Section 8). This bound directly holds for our harder, asynchronous problem variant.

In our lower bound, for given values of γ, g_0 and n, we design an input distribution that (1) sets g to g_0/γ with probability $1/2$, and sets g to γg_0 otherwise; (2) distributes g good jobs uniformly but randomly throughout the input; and (3) creates an estimator that effectively hides both the value of g and also the location of the good jobs, and always has estimation gap at most γ on the input (see Section 8.1 and Lemma 16 for details).

The key technical challenge is to set up a matrix in order to use Yao's minimax principle [123] to prove a lower bound on the expected cost for any randomized algorithm. Yao's Principle allows us to reframes a randomized algorithm as a probability distributions over deterministic algorithms. Instead of analyzing the expected performance of a randomized algorithm on the worst-case input, it considers the worst deterministic algorithm against a carefully chosen distribution over inputs. The key insight is that the best randomized algorithm cannot outperform the best deterministic algorithm

against this "hard" input distribution, allowing one to prove lower bounds by designing such distributions rather than reasoning directly about all random choices.

In particular, we need a game-theory payoff matrix where the rows are deterministic algorithms, and the columns are deterministic inputs. A natural approach is for the matrix entries to be the ratios $A/\sqrt{Bg}$, where A is the cost to our algorithm. Unfortunately, this fails to achieve our goal since this would yield $E[A/\sqrt{Bg}]$ and we need $E[A]/E[\sqrt{Bg}]$, which are not generally equal. Some lower bound proofs can circumvent this type of problem. For example, lower bound proofs on competitive ratios of online algorithms circumvent the problem because the offline cost does not vary with the choice of the deterministic algorithm, i.e. from row to row. So the expected offline cost can be factored out and computed separately (see [16], Chapter 5). Unfortunately, this does not work for us since our random variables A and B are interdependent, and vary across both rows and columns.

To address this challenge, we first set the matrix entries to $E(A) - \sqrt{E(B)g\gamma})$, and then prove that the minimax for such a payoff matrix has expected value at least 0. Then, we use Jensen's inequality and linearity of expectation to show that the minimax is also at least 0 when the matrix entries are $E(A - \sqrt{Bg\gamma})$ (see Lemma 17). Next, using Yao's principle, we can show the maximin is nonnegative, and thereby obtain a key inequality for A and B. Once we have this inequality, we can use algebra to establish our lower bound for any randomized algorithm (see Theorem 3).

We believe that this new approach — using Yao's principle on a matrix whose entries are $X - Y$ in order to bound the ratio $E(X)/E(Y)$, and thereby prove some target lower bound of $E(Y)$ — may have broader applications to proving lower bounds for resource competitive algorithms (see our expanded discussion of related work in Section 2).

5. An Estimator with Broad Application

Our main goal is to design algorithms that harness the power of estimators, rather than to create the estimators themselves. However, to convey the broad applicability of our results, we consider various distributions for the arrival of good jobs that correspond with real-world observations and established traffic models. We describe an example estimator, with the properties defined in Section 1.1, which we show has small γ for many types of dis-

tributions. We then illustrate executions of both LINEAR and LINEAR-POWER using this estimator.

5.1. Poisson Estimator

Assume good jobs are generated via a Poisson distribution with parameter λ, so that the expected interval length between good jobs is $1/\lambda$. Poisson job arrivals are one of the most commonly used statistical models, and are consistent with empirical findings [83, 21].

For simplicity, we assume exact knowledge of the Poisson parameter λ, but a constant factor approximation suffices for the analysis below. For an interval $\mathcal{I}$, let $\ell(\mathcal{I})$ be the length of $\mathcal{I}$. Then, for any interval $\mathcal{I}$, set

$$\hat{g}(\mathcal{I}) = \ell(\mathcal{I})/\lambda$$

5.1.1. Bounding $\boldsymbol{\gamma}$

We now show that w.h.p. in g, i.e. probability at least $1 - 1/g^k$ for any positive k, $\gamma = O(\log g)$ for the Poisson estimator. This holds for *all* distributions of bad jobs including: no bad jobs; more bad jobs than good; bad jobs uniformly distributed; and bad jobs that are clustered in any manner.

As a special case, consider when the adversary inserts bad jobs using the same Poisson distribution as the good jobs. Then, it is impossible to differentiate a good job from a bad job with better than a (fair) coin toss. Thus, the estimation gap remains $O(\log g)$ even though we can not classify jobs as good or bad. So, estimation can be easier than classification.

Analysis. To formally bound γ, consider any input distribution with g good jobs generated by the Poisson process. We show that w.h.p. in g, $\hat{g}$ ensures that $\gamma = O(\log g)$.

Recall that $g(\mathcal{I})$ is the number of good jobs generated during $\mathcal{I}$. We rely on the following fact which holds by properties of the Poisson process, Chernoff and union bounds. In the following w.h.p. means with probability of error that is polynomially small in g.

Fact 1. *For some constant C, w.h.p. for any interval $\mathcal{I}$:*

1. *If $g(\mathcal{I}) < C \ln g$, then $\ell(\mathcal{I}) = O((1/\lambda) \log g)$*
2. *If $g(\mathcal{I}) \geq C \ln g$, then $\ell(\mathcal{I}) = \Theta((1/\lambda) g(\mathcal{I}))$.*

We can now show the following simple fact.

Fact 2. *With high probability, the estimator $\hat{g}$ has estimation gap $\gamma = O(\log g)$.*

Proof. Consider two cases:

Case: $g(\mathcal{I}) < C \ln g$. Then, by Fact 1 (1), $\ell(\mathcal{I}) = O((1/\lambda)\log g)$, and so $\hat{g}(\mathcal{I}) = O(\log g)$. Thus, there exists some γ, $\gamma = O(\log g)$ such that

$$g(\mathcal{I}) - \gamma \leq \hat{g}(\mathcal{I}) \leq g(\mathcal{I}) + \gamma$$

Hence, the estimation gap on this type of interval is $O(\log g)$.

Case: $g(\mathcal{I}) \geq C \ln g$. Then by Fact 1 (2), w.h.p., $\ell(\mathcal{I}) = \Theta((1/\lambda)g(\mathcal{I}))$, and so $\hat{g}(\mathcal{I}) = \Theta(g(\mathcal{I}))$. So, there exists some γ, $\gamma = O(1)$ such that

$$(1/\gamma)g(\mathcal{I}) \leq \hat{g}(\mathcal{I}) \leq \gamma g(\mathcal{I})$$

Hence, the estimation gap on this type of interval is $O(1)$. We conclude that for this example input distribution, w.h.p., $\hat{g}$ is an estimator with $\gamma = O(\log g)$. □

Estimating λ. A simple way to estimate λ is to use job arrival data from a similar day and time. To handle possible spikes in usage by clients, one may need to analyze data on current jobs, including: packet header information [22], geographic client location [48], timing of requests [24], and spikes in traffic rate [73].

5.2. An Estimator for Weibull, Gamma, and Distributions with Exponentially Bounded Moments

The $\gamma = O(\log g)$ result for the Poisson distribution can be generalized to arrival times given by other distributions whose moments do not grow super-exponentially.

In particular, for any $i \geq 1$, let X_i be a random variable giving the time elapsed between the $i-1$ and i-th arrival. Assume that the X_i values are independent and identically distributed. Then, let $Y_i = X_i - E(X_i)$, and assume that for some real positive L and every positive integer k,

$$E(|Y_i^k|) \leq \frac{1}{2}E(Y_i^2)L^{k-2}k!$$

In other words, the above says that the k-th moment of Y_i is bounded by an exponential in k. This is true for common arrival time processes such as the Weibull [20] and the gamma distributions [32] with fixed parameters.

Then set $\rho = 1/E(X_i)$, and we can use an estimator that for any interval $\mathcal{I}$, sets

$$\hat{g}(\mathcal{I}) = \rho\ell(\mathcal{I})$$

We can again show the following.

Fact 3. *With high probability, the estimator $\hat{g}$ has estimation gap $\gamma = O(\log g)$.*

To show this, we use Bernstein's inequality (specifically, the second inequality in [120]; see also [108]) to prove a result analogous to Fact 1, but with $1/\lambda$ replaced by μ. Then, the result follows directly.

5.3. Adversarial Queuing Theory Estimator

To capture a range of network settings, adversarial queuing theory (AQT) [17] parameterizes the behavior of packet arrivals by an injection rate, ρ where $0 < \rho \leq 1$, and a non-negative integer, $\boldsymbol{b}$, that denotes burstiness. In AQT, a common arrival model is the ***leaky-bucket adversary***[36] where, for any t consecutive slots, at most $\rho t + b$ packets may arrive, for some number b (for examples, see [50, 49, 6, 31, 5]). The physical metaphor is that there is a bucket of size b gallons. Water enters the bucket according to the (adversarial) arrival process and it leaves at a constant rate of ρ. The model requires that b be large enough so that the bucket never overflows.

We define a new variant, the ***rain-barrel model***, that both lower bounds and upper bounds the number of arriving good jobs. In particular, we require that: for any interval of t seconds, between $(1/b)\rho t - b$ and $b\rho t + b$ good jobs arrive. The physical metaphor is that there is a barrel of size b and an additional reservoir of b gallons that can be added to the barrel at any time. Water enters the barrel according to the (adversarial) arrival process and leaves at a tunable rate in the range $\rho \cdot [1/b, b]$. The new requirement is that there is always water available, but that the barrel never overflows.

In such a model, with knowledge of ρ, it is easy to design an estimator with estimation gap $\gamma = b$. In particular, again set $\hat{g}(\mathcal{I}) = \rho\,\ell(\mathcal{I})$. By the definition of the bounded bucket model, this estimator has estimation gap $\gamma = b$.

Notably, the value b need not be known a priori. Further, as with the previous estimators, using a constant factor approximation to ρ will not change the asymptotic value of γ.

5.4. Zero Latency and Asynchronous Example Runs

We now illustrate the execution of our zero-latency and asynchronous algorithms under Poisson arrivals, using our first estimator (Section 5.1); results are similar for our other two estimators.

Zero-Latency. First, suppose the estimator is perfectly accurate. Then, $\hat{g}(\mathcal{I})$ is 0 during an iteration until a good job arrives. Once a good job arrives, $\hat{g}(\mathcal{I})$ increases from 0 to 1, and a new iteration begins, starting with this new good job. The value PRICE is set back to 1, and the good job pays a fee of 1. Then, the cost of each good job is 1 and the cost of each bad job is the number of bad jobs received since the last good job was received. There are exactly g iterations. If b bad jobs are submitted, the adversary's costs are minimized when the bad jobs are evenly spread over the iterations, so the cost for each bad job ranges from 1 to b/g in each of the g iterations, for a total cost of $B = \Theta(b^2/g)$.

Now, consider the special case where the arrival of good jobs is a Poisson process with parameter λ, and we use the example estimator from Section 5.1. Figure 3 illustrates this scenario. Here, iterations are demarcated by multiples of $1/\lambda$ and PRICE is reset to 1 at the end of each iteration. Many bad jobs may be present over these iterations, and there may be more than 1 good job per iteration (as depicted).

The lengths of time between good jobs are independently distributed, exponential random variables with parameter λ (see [88], Chapter 8) that is, the probability that j good jobs occur in time t is $e^{(-t\lambda)}\frac{(t\lambda)^j}{j!}$. The adversary can decide where to place bad jobs. We now consider what happens with LINEAR. When using the example estimator from Section 5.1, for all $i \geq 1$, the i-th iteration ends at time i/λ, and the current price is reset to 1.

1. If there are no bad jobs, the expected cost of the i-th iteration to the algorithm is $O(X_i^2)$ where X_i is the number of good jobs in a period of length $1/\lambda$, and $\sum_i X_i = g$. Since $E[X_i^2] = \sum_j j^2 e^{(-\lambda/\lambda)}\frac{(\lambda/\lambda)^j}{j!}$, the expected cost to the algorithm for any iteration i is $(1/e)\sum_j j^2(1/j!) = O(1)$.

2. Fix any iteration i. By paying $\Omega(b_i^2)$, the adversary can post b_i bad jobs early on, thereby forcing each good job to pay a fee of at least $b_i + 1$. The algorithm also pays a service cost of $b_i + 1$.

Asynchronous. With communication latency between the server and the clients, jobs may submit multiple fees over time. If the submitted solution

Figure 3: (Left) LINEAR with iterations of length $1/\lambda$, using the estimator from the zero-latency example in Section 5.4. Black circles are good jobs, red circles are bad. Since the estimation gap is small, each iteration contains about 1 good job. (Right) Blowup of the last iteration, with b_i bad jobs followed by 1 good job. The adversary's cost is $B_i = \Omega(b_i^2)$ and the algorithm's cost is $A_i = O(b_i)$, as described in Section 5.4.

hardness is less than the current price, the job request is denied. When this happens, we say that the corresponding job is ***bounced*** (see Figure 4). Alternatively, if the fee exceeds the price, then the job overpays.

Now, assume our example estimator, so that iterations end at the same time as the zero-latency case above. Further, assume a simple distribution for jobs where exactly 1 job is generated per iteration at some uniformly distributed time.

Note that M is determined by the maximum latency Δ, and λ. If the latency Δ is much larger than $1/\lambda$, then the adversary can delay messages so that $M = \Delta/\lambda$ good jobs, bounced from the previous Δ/λ iterations, are received in the current iteration. The first $M/2$ of these jobs are serviced, driving up the value of PRICE to at least $M/2$. During this time, the remaining jobs are bounced up to $\log(M/2)$ times. Thus, the algorithm's total cost is $\Omega((g/M)(M^2)) = \Omega(gM)$.

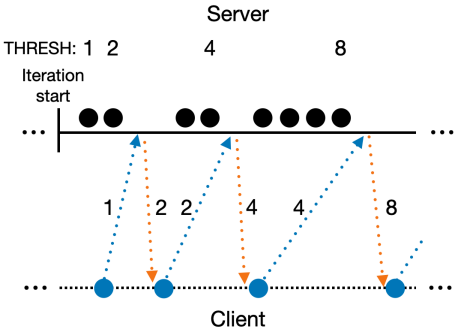

Figure 4: The blue circle is a job that is repeatedly bounced, the black circles are other serviced jobs. Arrows are communication from the server to the client of the PRICE value, and from the client to the server of fees. The blue job is bounced 3 times and pays $1 + 2 + 4 + 8$.

For these jobs $i = 2, ..., (M-1)$, the adversary can cause the i-th job to bounce $\lfloor \lg(M-i) \rfloor$ times with payments of 2^j, for $j = 0, ..., \lfloor \lg(M-i) \rfloor$ in the current iteration. Thus, the total cost to the algorithm is $\Omega((g/M)(M^2)) = \Omega(gM)$.

6. Analysis of LINEAR

In this section, we formally analyze LINEAR. To give intuition about the choice of cost function, we analyze a general class of algorithms that define iterations like LINEAR: algorithms that charge the i-th job in the iteration a fee of i^α, for some positive α. Our analysis shows that LINEAR, which sets $\alpha = 1$, is asymptotically optimal across this class of algorithms.

Let b be the total number of bad jobs, and recall that g is the number of good jobs. We note that all good jobs pay a cost of at most 1 before they learn the current value of PRICE, which incurs a total cost of $O(g)$. This is asymptotically equal to the total service cost. Thus, for simplicity, in this section, we only count the costs for the fees submitted when the jobs are serviced. Throughout, we let $\log(\cdot)$ denote the logarithm base 2.

Lemma 1. *Let ℓ be the number of iterations. Then $\ell \leq \gamma(g+1)$.*

Proof. Let ℓ be the total number of iterations. For all j, $1 \leq j \leq \ell$, let $\mathcal{I}_j$ be the time interval spanning the j-th iteration, i.e the interval $(t_0, t_1]$ where t_0 is the time the j-th iteration begins and t_1 is the time the j-th iteration ends.

By the algorithm, $\hat{g}(\mathcal{I}_j) \geq 1$ for all j. Let $\mathcal{I}_{tot} = \cup_{1 \leq j \leq \ell} \mathcal{I}_j$. Using the additive property of $\hat{g}$, we get $\hat{g}(\mathcal{I}_{tot}) = \sum_{1 \leq j \leq \ell} \hat{g}(\mathcal{I}_j) \geq \ell$. But by our definition of γ, we know that $\hat{g}(\mathcal{I}_{tot}) \leq \gamma(g(\mathcal{I}_{tot}) + 1) = \gamma(g+1)$. Thus, $\ell \leq \gamma(g+1)$. □

Lemma 2. *The number of good jobs serviced in any iteration is no more than $\gamma(\gamma+1)+1$.*

Proof. Fix some iteration that starts at time t_0 and ends at time t_1. Let $\mathcal{I} = [t_0, t_1]$ be the interval associated with that iteration. Let $\mathcal{I}' = [t_0, t_1']$, where t_1', $t_0 \leq t_1' \leq t_1$, is a time after which the server has received the second to last job request in the iteration, but not yet the last job request. First, by construction of $\mathcal{I}'$, we have

$$g(\mathcal{I}') \geq g(\mathcal{I}) - 1 \tag{1}$$

Then, by the γ estimation gap for $\hat{g}$ on the input, we know that $g(\mathcal{I}') \leq \gamma(\hat{g}(\mathcal{I}') + \gamma)$. Plugging in the fact that $\hat{g}(\mathcal{I}') < 1$, which holds by the properties of LINEAR, we get:

$$g(\mathcal{I}') < \gamma + \gamma^2 \tag{2}$$

Putting together equations 1 and 2 we get that $g(\mathcal{I}) \leq \gamma + \gamma^2 + 1$. □

Lemma 3. *Let ℓ be the number of iterations. Then, $\ell \geq \max\left(1, \frac{g-\gamma^2}{3\gamma^4}\right)$.*

Proof. Let $\mathcal{I}_j$ be the interval spanning iteration j for $1 \leq j \leq \ell$, where ℓ is the total number of iterations. By Lemma 2, $g(\mathcal{I}_j) \leq \gamma(\gamma+1)+1$ for all j. By the estimation gap for $\hat{g}$ on the input, we get $\hat{g}(\mathcal{I}_j) \leq \gamma g(\mathcal{I}_j) + \gamma$. Combining these two inequalities, we get $\hat{g}(\mathcal{I}_j) \leq \gamma^3 + \gamma^2 + 2\gamma$.

Let $\mathcal{I}_{tot} = \cup_{1\leq j\leq \ell}\mathcal{I}_j$. Using the additive property of $\hat{g}$, we get

$$\hat{g}(\mathcal{I}_{tot}) = \sum_{1\leq j\leq \ell} \hat{g}(\mathcal{I}_j) \leq (\gamma^3+\gamma^2+2\gamma)\ell.$$

Also, by the estimation gap for $\hat{g}$ on the input, we have $\hat{g}(\mathcal{I}_{tot}) \geq g(\mathcal{I}_{tot})/\gamma - \gamma$. Thus, if $\gamma^3+\gamma^2+2\gamma > 0$, which must hold since $\gamma > 0$, we have:

$$\ell \geq (g/\gamma - \gamma)/(\gamma^3+\gamma^2+2\gamma) = (g-\gamma^2)/\gamma(\gamma^3+\gamma^2+2\gamma)$$

This establishes the lemma statement. □

Recall that B is the cost to the adversary.

Lemma 4. *For any positive α, $B \geq \frac{(b-\gamma(g+1))^{\alpha+1}}{(\alpha+1)(\gamma(g+1))^{\alpha}}$.*

Proof. The cost to the adversary is minimized when the bad jobs in each iteration occur before the good jobs. Furthermore, this cost is minimized when the bad jobs are distributed as uniformly as possible across iterations. To see this, assume j jobs are distributed non-uniformly among ℓ iterations. Thus, there exists at least one iteration i_1 with at least $\lceil j/\ell \rceil + 1$ bad jobs, and also at least one iteration i_2 with at most $\lfloor j/\ell \rfloor - 1$. Since the cost function i^α is monotonically increasing, moving one bad job from iteration i_1 to iteration i_2 can only decrease the total cost.

To bound the adversarial cost in an iteration, we use an integral lower bound. Namely, for any x and α, $\sum_{i=1}^{x} i^\alpha \geq 1 + \int_{i=1}^{x} i^\alpha \, di \geq \frac{x^{\alpha+1}}{\alpha+1}$, where the last inequality holds for $\alpha \geq 0$. If the number of iterations is $\ell > 0$, we let

$x = \lfloor b/\ell \rfloor$ in the above and bound the total cost of the adversary as:

$$\begin{aligned} B &\geq \frac{\ell}{\alpha+1}\left\lfloor \frac{b}{\ell} \right\rfloor^{\alpha+1} \\ &\geq \frac{\ell}{\alpha+1}\left(\frac{b}{\ell}-1\right)^{\alpha+1} \\ &\geq \frac{b-\ell}{\alpha+1}\left(\frac{b}{\ell}-1\right)^{\alpha} \\ &\geq \frac{b-\gamma(g+1)}{\alpha+1}\left(\frac{b-\gamma(g+1)}{\gamma(g+1)}\right)^{\alpha} \\ &\geq \frac{(b-\gamma(g+1))^{\alpha+1}}{(\alpha+1)(\gamma(g+1))^{\alpha}} \end{aligned}$$

The second step holds since $(b/\ell) - 1 = (b-\ell)/\ell$. The fourth step holds since, by Lemma 1, $\ell \leq \gamma(g+1)$. □

Recall that the cost to the total algorithm is the sum of the fees paid by clients and the service cost. In the following, let $\boldsymbol{A_F}$ denote the fees paid by the clients.

Lemma 5. *For $\alpha \geq 1$,*

$$A_F = O\left(\gamma^{2\alpha} g + \gamma^2 b^{\alpha}\right).$$

For $0 \leq \alpha < 1$,

$$A_F = O\left(\gamma^{2\alpha+4} g + \frac{\gamma^{4\alpha} b^{\alpha+1}}{\max\{1, (g-\gamma^2)^{\alpha}\}}\right).$$

Proof. Let ℓ be the last iteration with some positive number of good jobs, and for $i \in [1, \ell]$ let g_i and b_i be the number of good and bad jobs in iteration i, respectively. Then, the fees paid in an iteration are maximized when all good jobs come at the end of the iteration. For a fixed iteration i, the cost is: $\sum_{j=1}^{g_i} (b_i + j)^{\alpha}$. Then, the total fees paid, A_F is:

$$\sum_{i=1}^{\ell} \sum_{j=1}^{g_i} (b_i + j)^{\alpha}$$

Note that for all $1 \leq i \leq \ell$, $g_i \leq \gamma^2 + \gamma + 1$, by Lemma 2. For simplicity of notation, in this proof, we let $\beta = \gamma^2 + \gamma + 1$; and note that for all $i \in [1, \ell]$, $g_i \leq \beta$.

We use exchange arguments to determine the settings for which A_F is maximized subject to our constraints. Consider any initial settings of $\overrightarrow{g}$ and $\overrightarrow{b}$. We assume, without loss of generality, that the b_i values are sorted in decreasing order, since if this is not the case, we can swap iteration indices to make it so, without changing the function output.

Good jobs always packed left. We first show that A_F is maximized when the good jobs are "packed left": β good jobs in each of the first $\lfloor g/\beta \rfloor$ iterations, and $g \bmod \beta$ good jobs in the last iteration.

To see this, we first claim that, without decreasing A_F, we can rearrange good jobs so that the g_i values are non-increasing in i. In particular, consider any two iterations j, k such that $1 \leq j < k \leq \ell$, where $g_j < g_k$. We move the last $g_k - g_j$ good jobs in iteration k to the end of iteration j. This will not decrease A_F since the cost incurred by each of these moved jobs can only increase, since $b_j \geq b_k$, and our RB-cost function can only increase with the job index in the iteration. Repeating this exchange establishes the claim.

Next, we argue that, without decreasing A_F, we can rearrange good jobs so that the g_i values are β for all $1 \leq i < \ell$. To see this, consider the case where there is any iteration before the last that has less than β good jobs, and let j be the leftmost such iteration with $g_j < \beta$. Since $j < \ell$, there must be some iteration k, where $j < k$, such that $g_k > 0$. We move the last good job from iteration k to the end of iteration j. This will not decrease A_F since the cost incurred by this moved job can only increase, since our RB-cost function can only increase with the job index in the iteration. Repeating this exchange establishes the claim.

Analyzing δ_i. Next, we set up exchange arguments for the bad jobs. For any integer x, define

$$\delta(x) = \sum_{j=1}^{\beta} (x + 1 + j)^{\alpha} - \sum_{j=1}^{\beta} (x + j)^{\alpha}$$

If we consider swapping a single bad job from iteration k into iteration i where $1 \leq i < k \leq \ell$, this results in the following change in the overall cost: $\delta(b_i) - \delta(b_k - 1)$.

Let $f : \mathbb{R} \to \mathbb{R}$ such that, $f(x) = \sum_{j=1}^{\beta} (x + j)^{\alpha}$. For any $i \in [1, \ell]$, this function is continuous when $x \in [b_i, b_i + 1]$ and differentiable when

$x \in (b_i, b_i + 1)$, since it is the sum of a finite number of continuous and differentiable functions respectively. For any $i \in [1, \ell]$, by applying the Mean Value Theorem for the interval $[b_i, b_i + 1]$, we have:

$$\delta(b_i) = \frac{f(b_i + 1) - f(b_i)}{(b_i + 1) - b_i} = f'(x_i) \text{ for some } x_i \in [b_i, b_i + 1].$$

Bad jobs packed left when $\alpha \geq 1$. When $\alpha \geq 1$, f is convex. Hence, $f'(x)$ is non-decreasing in x. Using this fact and the Mean Value Theorem analysis above, we get that $\delta(b_i) \geq f'(b_i)$; and, for any $i < k \leq \ell$, $\delta(b_k - 1) \leq f'(b_k)$. In addition, $f'(b_i) \geq f'(b_k)$ since $b_i \geq b_k$, by our initial assumption that the b_i values are non-increasing in i. Thus, $\delta(b_i) - \delta(b_k - 1) \geq 0$.

This implies that A_F does not decrease whenever we move a bad job from some iteration with index $k > 1$ to iteration $i = 1$. Hence, A_F is maximized when all bad jobs occur in the first iteration.

Bad jobs evenly spread when $\alpha < 1$. When $\alpha < 1$, f is concave. Hence, $f'(x)$ is non-increasing in x. Using this fact and the Mean Value Theorem analysis above, we get that $\delta(b_i) \leq f'(b_i)$; and, for any $1 < k \leq \ell$, $\delta(b_k - 1) \geq f'(b_k)$. In addition, $f'(b_k) \geq f'(b_i)$ since $b_i \geq b_k$, by our initial assumption that the b_i values are non-increasing in i. Thus, $\delta(b_i) - \delta(b_k - 1) \leq 0$.

This implies that A_F increases whenever we swap bad jobs from an iteration with a larger number of bad jobs to an iteration with a smaller number. Hence, A_F is maximized when bad jobs are distributed as evenly as possible across iterations.

Upper bound on A_F. We can now bound A_F via a case analysis.

Case $\alpha \geq 1$: With the above exchange argument in hand, we have:

$$\begin{aligned}
A_F &\leq \left(\frac{g}{\beta}\right) \sum_{j=1}^{\beta} j^\alpha + \sum_{j=1}^{\min(\beta, g)} (b + j)^\alpha \\
&\leq \left(\frac{g}{\beta}\right) \sum_{j=1}^{\beta} j^\alpha + \min(\beta, g)(b + \beta)^\alpha \\
&\leq 2\left(\frac{g}{\beta}\right) \frac{\beta^{\alpha+1}}{\alpha + 1} + \min(\beta, g)(b + \beta)^\alpha \\
&\leq 2\left(\frac{g}{\beta}\right) \frac{\beta^{\alpha+1}}{\alpha + 1} + \beta(2b)^\alpha + g(2\beta)^\alpha \\
&= O\left(\beta^\alpha g + \beta b^\alpha\right)
\end{aligned}$$

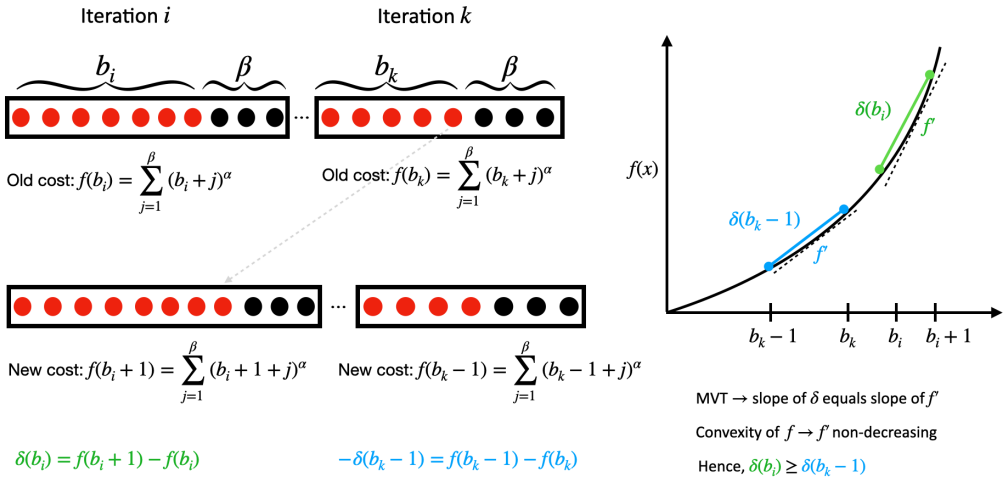

Figure 5: Illustration of an exchange of one bad job (a red ball) from iteration k to iteration i, with all good jobs (black balls) remaining in place. The change in cost for iteration i is $\delta(b_i)$, and change in cost for iteration k is $\delta(b_k - 1)$. The Mean Value Theorem (MVT) says that for any integer x, the value of $\delta(x)$ equals $f'(y)$ for some $y \in [x, x+1]$. Using the convexity of f' (for $\alpha \geq 1$), this implies that $\delta(b_i) \geq \delta(b_k - 1)$. This means that the exchange shown can only increase the algorithmic cost.

where the third line follows by upper bounding the sum by an integral, and the fourth line follows from the inequality $(x+y)^d \leq (2x)^d + (2y)^d$ for any positive x, y, and d, which holds because $x + y \leq 2x$ or $x + y \leq 2y$. Finally, note that replacing β with $\gamma^2 + \gamma + 1 \leq 3\gamma^2$ gives the same asymptotic upper bound in the lemma statement.

Case $\alpha < 1$: With the above exchange argument in hand, we have:

$$\begin{aligned} A_F &\leq \ell \sum_{j=1}^{\beta} (b/\ell + j)^\alpha \\ &\leq \ell \sum_{j=1}^{\beta + b/\ell + 1} j^\alpha \\ &\leq 2\ell \frac{(\beta + b/\ell + 1)^{\alpha+1}}{\alpha + 1} \end{aligned}$$

$$
\begin{aligned}
&\leq 2\ell \frac{(2\beta+2)^{\alpha+1} + (2b/\ell)^{\alpha+1}}{\alpha+1} \\
&= O\left(\gamma^{2\alpha+3} g + \frac{\gamma^{4\alpha} b^{\alpha+1}}{(\alpha+1)\max\{1,(g-\gamma^2)^{\alpha}\}}\right) \\
&= O\left(\gamma^{2\alpha+3} g + \frac{\gamma^{4\alpha} b^{\alpha+1}}{\max\{1,(g-\gamma^2)^{\alpha}\}}\right).
\end{aligned}
$$

In the fourth line above, we again use the fact that $(x+y)^d d \leq (2x)^d + (2y)^d$, since either $x \geq y$ or $y \geq x$. In the fifth line we use the fact that $\beta \leq 3\gamma^2$ and we use Lemma 1 to upper bound the ℓ in the numerator, and use Lemma 3 to lower bound the ℓ in the denominator. In the last line, we use the fact that $1/\alpha$ is a fixed constant. □

Note from Lemma 5 that for $b \geq \gamma^2$, A_F is minimized when $\alpha \geq 1$. Using this fact, we show $\alpha = 1$ minimizes the total cost of the algorithm as a function of B and g. The following theorem bounds the algorithm's cost.

Lemma 6. *The total cost to* LINEAR *is* $O\left(\gamma^{5/2}\sqrt{B(g+1)} + \gamma^3(g+1)\right)$.

Proof. To bound the cost of A, we perform a case analysis.

Case: $\boldsymbol{b \geq 2\gamma(g+1)}$. In this case, $\gamma(g+1) \leq b/2$. By Lemma 4, we have:

$$
B \geq \frac{(b-\gamma(g+1))^{\alpha+1}}{(\alpha+1)(\gamma(g+1))^{\alpha}} \geq \frac{(b/2)^{\alpha+1}}{(\alpha+1)(\gamma(g+1))^{\alpha}} = \frac{b^{\alpha+1}}{2^{\alpha+1}(\alpha+1)\gamma^{\alpha}(g+1)^{\alpha}}
$$

The service cost is $b + g \leq 2b$, since in this case $g \leq b/(2\gamma) \leq b/2$. Let A denote the total cost to the algorithm. We now use Lemma 5; specifically, the case for $\alpha \geq 1$, which gives the smallest worst case total fees paid, across all values of g and b and γ. This yields:

$$
A \leq 2b + c(\gamma^{2\alpha} g + \gamma^2 b^{\alpha})
$$

where $c > 0$ is a sufficiently large constant. Thus we have:

$$
\begin{aligned}
A &\leq 2b + c(\gamma^{2\alpha} g + \gamma^2 b^{\alpha}) \\
&\leq 2b + c(\gamma^{2\alpha}(b/2\gamma) + \gamma^2 b^{\alpha}) \\
&\leq 2b + c(\gamma^{2\alpha-1} b + \gamma^2 b^{\alpha})
\end{aligned}
$$

Where the second line holds since $\gamma(g+1) \leq b/2$ implies that $\gamma g \leq b/2$, or $g \leq b/(2\gamma)$. Note that the above is minimized over the range $\alpha \geq 1$, when $\alpha = 1$. In this case, we have:

$$A \leq 2b + c\left(\gamma b + \gamma^2 b\right) \leq (c\gamma + \gamma^2 + 2)b = O\left(\gamma^{5/2}\sqrt{B(g+1)}\right).$$

In the equality above, for $\alpha = 1$, $B \geq \frac{b^2}{8\gamma(g+1)}$, which implies that $b \leq \sqrt{8\gamma B(g+1)}$.

Case: $b < 2\gamma(g+1)$. In this case, the service cost is $g+b \leq (2\gamma+1)(g+1)$. So when $\alpha = 1$, we can bound the total cost as:

$$\begin{aligned} A &\leq (2\gamma+1)(g+1) + c(\gamma^2 g + \gamma^2 b) \\ &\leq (2\gamma+1)(g+1) + c(\gamma^2 g + \gamma^2(2\gamma(g+1))) \\ &= O(\gamma^3(g+1)) \end{aligned}$$

which completes the proof. □

The communications bounds for LINEAR are straightforward.

Lemma 7. *In* LINEAR*, the server sends $O(g)$ messages to clients, and clients send a total of $O(g)$ messages to the server. Also, each good job is serviced after an exchange of at most* 3 *messages between the client and server*

Proof. Since there is no latency, each client immediately receives the current value of PRICE if its first message to the server has a fee with value too small. Then, on the second attempt the client attaches a fee with the correct value, and so receives service. Hence, each good job is serviced with at most 2 messages from client to server and at most 1 message from server to client. □

Theorem 1 now follows directly from Lemmas 6 and 7.

7. Analysis of LINEAR-POWER

How much do the total fees paid increase for LINEAR-POWER in our harder asynchronous problem variant (recall Section 1.1)? We say that a good job "overpays" if the fee submitted when the job is serviced exceeds the price that the server demands. We bound the total amount overpaid by all good jobs in the asynchronous model. When compared to LINEAR,

the adversary's cost in LINEAR-POWER decreases by at most a factor of 2; and the total amount spent by good jobs that do not overpay increases by at most a factor of 2. Thus, overpayment is the only possible source of asymptotic increase in algorithmic costs.

Thus we upper bound the overpayment amount. To do this, we partition time into epochs. An ***epoch*** is a contiguous time interval ending with a period of 2Δ seconds, where the current price during the entire period never increases.

In the following, for all $i \geq 1$, let B_i be the amount spent by the adversary in epoch i. For convenience, we define $B_0 = 0$.

Lemma 8. *For all $i \geq 1$, the maximum current price in epoch i is at most*

$$3\gamma^2 \max\left(1, 2\sqrt{B_i}\right) \leq 6\gamma^2\sqrt{B_i + 1}$$

Proof. Fix an epoch $i \geq 1$ and let 2^x be the maximum current price in that epoch.

If $B_i > 0$, and the epoch consists solely of bad jobs, we know that $B_i \geq 2^{2(x-1)}$, because of the way that our server increases the current price. Hence, the bad jobs contribute no more than $((1/2)\log B_i) + 1$ to the value x, when $B_i > 0$.

In any period where the current price monotonically increases, the number of good jobs is no more than the number of good jobs in an iteration, which is at most $\gamma(\gamma + 1) + 1 \leq 3\gamma^2$ by Lemma 2. Thus, the total increase in x from good jobs is at most $\log(3\gamma^2)$.

Putting these together, we get that when $B_i = 0$, $x \leq \log 3\gamma^2$ and when $B_i > 0$, $x \leq ((1/2)\log B_i) + 1 + \log 3\gamma^2$. Hence the maximum current price is at most $3\gamma^2 \max(1, 2\sqrt{B_i}) \leq 6\gamma^2\sqrt{B_i + 1}$. □

Lemma 9. *For all $i \geq 1$, at most $6M(\log(\gamma(B_i + 1)) + 1)$ good jobs arrive in epoch i.*

Proof. Over every 2Δ seconds, starting from the beginning of the epoch, the current price must double, or else the epoch will end. Thus, the epoch lasts at most $2\Delta(x + 1)$ seconds, where 2^x is the maximum current price achieved in that epoch.

By Lemma 8, $2^x \leq 6\gamma^2\sqrt{B_i + 1}$; taking the log and simplifying, we get $x \leq 3 + 2\log(\gamma(B_i + 1))$. The lemma follows since over a period of 2Δ seconds, at most $2M$ good jobs can arrive. □

Lemma 10. *Any epoch has overpaying jobs only if either the epoch or its preceding epoch spans the beginning of some iteration.*

Proof. All jobs entering prior to the last 2Δ seconds of an epoch are serviced before the end of the epoch, since, by definition, the current price is non-increasing during those 2Δ seconds. Thus, all jobs serviced in some fixed epoch either have entered during that epoch or entered in the final 2Δ seconds of the preceding epoch.

If neither the current epoch nor the preceding epoch spanned the beginning of an iteration, then the current price never resets to 1 during the lifetime of any of these jobs. So the current price is non-decreasing during each such job's lifetime. Thus, none of the jobs can ever have an attached fee that is larger than the current price, and so no overpayment occurs. □

Lemma 11. *For all $i \geq 1$, the total amount that good jobs overpay in epoch i is at most $18M\gamma^2(\sqrt{B_{i-1}} + \sqrt{B_i} + 1)(\log(\gamma(B_i + 1)) + 2)$.*

Proof. First, note that every good job serviced in epoch i either arrived in epoch i: at most $6M(\log(\gamma(B_i + 1) + 1)$ such jobs by Lemma 9; or arrived in the last 2Δ seconds of the preceding epoch: at most $2M$ such jobs. Thus, there are at most $6M(\log(\gamma(B_i + 1)) + 2)$ good jobs serviced in epoch i.

By Lemma 8, the maximum current price in either epoch i or $i - 1$ is:

$$3\gamma^2 \max\left\{\sqrt{B_{i-1}}, \sqrt{B_i}, 1\right\} \leq 3\gamma^2\left(\sqrt{B_{i-1}} + \sqrt{B_i} + 1\right)$$

This is the maximum amount that any job serviced in epoch i can overpay. Multiplying by the number of good jobs serviced, we obtain the bound in the lemma statement. □

Lemma 12. *The total amount overpaid by good jobs in* LINEAR-POWER *is*

$$O\left(M\log(\gamma(B+2))\left(\gamma^{5/2}\sqrt{(g+1)B} + \gamma^3(g+1)\right)\right)$$

Proof. Let X be the total amount overpaid by good jobs in LINEAR-POWER. We will give two upperbounds on X and then combine them.

First, by Lemma 8 and the fact that for all $i \geq 1$, $B_i \leq B$, we know that the maximum amount any good job can overpay is at most $6\gamma^2\sqrt{B+1}$. Multiplying by all good jobs, we get:

$$X \leq 6(g+1)\gamma^2\sqrt{B+1} \tag{3}$$

Next, let S be the set of indices of epochs that have overpaying jobs. Then, by Lemma 11:

$$\begin{aligned}
X &\leq \sum_{j \in S} 18M\gamma^2(\sqrt{B_{i-1}} + \sqrt{B_i} + 1)(\log(\gamma(B_i + 1)) + 2) \\
&\leq 18M\gamma^2(\log(\gamma(B+1)) + 2) \sum_{j \in S}(\sqrt{B_{j-1}} + \sqrt{B_j} + 1) \\
&\leq 18M\gamma^2(\log(\gamma(B+1)) + 2) \left(|S| + \sum_{j \in S} \sqrt{B_{j-1}} + \sum_{j \in S} \sqrt{B_j}\right) \\
&\leq 18M\gamma^2(\log(\gamma(B+1)) + 2) \left(|S| + 2\sqrt{B}\left(\min(\sqrt{S}, \sqrt{B})\right)\right)
\end{aligned}$$

In the above, the fourth step holds by noting that the sum of the B_{j-1} and B_j terms are both at most B, applying Cauchy-Schwartz to both sums, and noting that the sum of the $\sqrt{B_i}$ can never be larger than B.

Next, noting that $|S| \leq 2\gamma(g+1)$ by Lemmas 1 and 10, we have for some constant C:

$$X \leq CM\gamma^2 \left(\gamma(g+1) + \sqrt{B}\left(\min(\sqrt{\gamma(g+1)}, \sqrt{B}\right) \log(\gamma(B+2)))\right) \quad (4)$$

Putting together equations 3 and 4, we have

$$X = O\left(\gamma^2\sqrt{B+1}\min\left(g+1, M\log(\gamma(B+2))\left(\min(\sqrt{\gamma(g+1)}, \sqrt{B+1})\right)\right)\right)$$

□

Lemmas 13, 14, and 15 below complete the proof of our second main result, Theorem 2.

Lemma 13. LINEAR-POWER *has total cost:*

$$\tilde{O}\left(\gamma^3\left(\sqrt{B+1}\min\left(g+1, M\sqrt{g+1}, M\sqrt{B+1}\right) + (g+1)\right)\right).$$

Proof. Fix the order in which jobs are serviced by LINEAR-POWER. The current price when a job is serviced in LINEAR-POWER will differ by at most a factor of 2 from LINEAR because of the fact that LINEAR-POWER uses powers of 2. Thus, the total amount spent by the adversary

decreases by at most a factor of 2, and the total amount spent by the algorithm on all jobs that do not overpay increases by at most a factor of 2. It follows that the asymptotic results in Theorem 1 hold, except for the costs due to overpayment by the good jobs.

So, we can bound the asymptotic cost of the algorithm as the cost from LINEAR plus the cost due to overpayment. The cost in the theorem statement is the asymptotic sum of the costs from Theorem 1 and from Lemma 12. This is:

$$O\Big(\gamma^2\sqrt{B+1}\min\Big(g+1, M\log(\gamma(B+2))\Big(\min(\sqrt{\gamma(g+1)}, \sqrt{B+1})\Big)\Big) + \gamma^{5/2}\sqrt{B(g+1)} + \gamma^3(g+1)\Big)$$

which is

$$\tilde{O}\left(\gamma^3\left(\sqrt{B+1}\min\left(g+1, M\sqrt{g+1}, M\sqrt{B+1}\right) + (g+1)\right)\right)$$

which completes the argument. □

Lemma 14. *The maximum time for any good job to start service is* $\Delta(4\log(\gamma(B+1))+4)$ *seconds.*

Proof. Fix any good job. Every time the job is bounced, the current price sent from the server to the corresponding client at least doubles. The client receives the new current price from the server within Δ seconds.

How many times can the current price change? If $B > 0$, the number of times that bad jobs can double the current price is at most $(1/2)\log B$. By Lemma 2, the number of good jobs in an iteration is at most $\gamma(\gamma+1)+1 \le 3\gamma^2$. Hence, the total number of times the current price can change for $B = 0$ is at most $2 + 2\log\gamma$; and for $B > 0$, at most $2 + 2\log\gamma + (1/2)\log B$. In both cases, this is at most $2 + 2\log\gamma + (1/2)\log(B+1) \le 2 + 2\log(\gamma(B+1))$.

Every time the job is bounced, the latency increases by at most 2Δ seconds: Δ seconds to send the job to the server, and Δ seconds to receive the new current price from the server. When the job is serviced it takes at most Δ seconds: the time to send the job to the server. Thus, the total time for the job to be serviced is at most $2\Delta(2\log(\gamma(B+1))+2) + \Delta \le \Delta(4\log(\gamma(B+1))+4)$. □

The following lemma bounds the number of messages sent from the server to clients and from clients to the server. Note that the adversary can cause the server to send a message to it for every request that it sends.

Lemma 15. *In* LINEAR-POWER *the server sends* $O(g \log(B + (3\gamma^2)))$ *messages to the clients and the clients send a total of* $O(g \log(B + 3\gamma^2))$ *messages to the server.*

Proof. Every time the server sends a message to the client, the current price sent to that client must at least double. How often can this happen? By Lemma 2 the number of good jobs in any iteration is at most $\gamma(\gamma+1)+1 \leq 3\gamma^2$. So, the total number of changes to the current price in any iteration is $\log(B + (3\gamma^2))$. So, the maximum current price ever achieved is at most $2^{\log(B+(3\gamma^2))}$. Hence, the maximum number of messages sent to any client is $O(\log(B + (3\gamma^2)))$. It follows that the total number of messages sent to clients is $O(g \log(B + (3\gamma^2)))$

Every time a client sends a new message to the server, the fee attached to that message doubles. The maximum current price set by the server over all iterations is at most $\log(B + 3\gamma^2)$, so a good job will send at most that number of messages. Hence, all g good jobs send a total of $O(g \log(B+3\gamma^2))$ messages to the server. □

8. Lower Bound Proof

In this section we give a lower bound for randomized algorithms for our problem zero-latency. This lower bound also holds directly for our harder asynchronous variant of our model.

The lower bound argument makes use of a particular estimator that satisfies additivity, and has estimation gap γ. The estimation gap γ is the only aspect our model measures for estimator performance. Hence, to prove a lower bound as a function of γ, the adversary can choose *any* estimator with estimation gap γ.

8.1. Our Adversary

Let γ be any positive integer, g_0 be any multiple of γ, and n be any multiple of $g_0\gamma$. To prove our lower bound, we choose both an input distribution and also a specific estimator.

Distribution. There are two possible distributions of n jobs. In both distributions, the jobs are evenly spread in a sequence over T seconds for any value T; in particular, the i-th job occurs at time $T(i/n)$.

In distribution one, the number of good jobs is $g_1 = \gamma g_0$. In distribution two, it is $g_2 = g_0/\gamma$. Our adversary chooses one of these two values, each with probability 1/2.

Then, for $i \in \{1, 2\}$, we partition from left to right the sequence of jobs into contiguous subsequences of length n/g_i. In each partition, we set one job, selected uniformly at random, to be good and the remaining jobs to be bad.

Estimator. For any interval $\mathcal{I}$, let $num(\mathcal{I})$ be the number of (good and bad) jobs overlapped by $\mathcal{I}$, i.e. $num(\mathcal{I}) = |\mathcal{I} \cap \{k(T/n) : k \in \mathbb{Z}^+\}|$. Then

$$\hat{g}(\mathcal{I}) = (g_0/n)num(\mathcal{I})$$

It is easy to verify that the above function has the additive property, and so is an estimator. Moreover, this estimator has two important properties. First, the estimation gap is always at most γ for either input distribution (see Lemma 16). Second, it is independent of the choice of distribution one or two.

8.2. Analysis

We first show that this estimator has γ estimation gap for both distributions.

Lemma 16. *For both distributions, for any interval $\mathcal{I}$, we have:*

$$g(\mathcal{I})/\gamma - \gamma \leq \hat{g}(\mathcal{I}) \leq \gamma g(\mathcal{I}) + \gamma$$

Proof. Let $g_{\max} = \max\{g_1, g_2\} = \gamma g_0$ and $g_{\min} = \min\{g_1, g_2\} = g_0/\gamma$, and fix any interval $\mathcal{I}$. Then, for each distribution we have the following based on the observation that there is one good job in every partition of size n/g_i for $i \in \{1, 2\}$

$$g(\mathcal{I}) \leq \lceil num(\mathcal{I})/(n/g_i) \rceil \leq \left(\frac{g_{\max}}{n}\right) num(\mathcal{I}) + 1 \leq \left(\frac{\gamma g_0}{n}\right) num(\mathcal{I}) + 1. \quad (5)$$

Thus:

$$\begin{aligned}\left(\frac{1}{\gamma}\right) g(\mathcal{I}) - \gamma &\leq \left(\frac{1}{\gamma}\right)\left(\frac{\gamma g_0}{n} num(\mathcal{I}) + 1\right) - \gamma \\ &\leq \left(\frac{g_0}{n}\right) num(\mathcal{I}) + \frac{1}{\gamma} - \gamma \\ &\leq \hat{g}(\mathcal{I})\end{aligned}$$

In the above, the first line follows from Equation 5, and the last line follows for $\gamma \geq 1$ and by the properties of our estimator.

For the other direction, note that for any interval $\mathcal{I}$, we have:

$$g(\mathcal{I}) \geq \lfloor num(\mathcal{I})/(n/g_i) \rfloor \geq \left(\frac{g_{\min}}{n}\right) num(\mathcal{I}) - 1 \geq \left(\frac{g_0}{\gamma n}\right) num(\mathcal{I}) - 1. \quad (6)$$

Thus:

$$\begin{aligned} \gamma g(\mathcal{I}) + \gamma &\geq \gamma \left(\frac{g_0}{\gamma n} num(\mathcal{I}) - 1\right) + \gamma \\ &\geq \left(\frac{g_0}{n}\right) num(\mathcal{I}) \\ &= \hat{g}(\mathcal{I}) \end{aligned}$$

In the above, the first line follows from Equation 6, and the last line follows from the fact that $\gamma \geq 1$, and the definition of the estimator. Hence, by the above analysis, our estimator has γ estimation gap for both distributions. □

Since the estimator returns the same outputs on both distributions, any deterministic algorithm will set job costs to be the same. This fact helps in proving the following lemma about expected costs.

Lemma 17. *Consider any deterministic algorithm that runs on the above adversarial input distribution and any estimator. Define the following random variables: A is the cost to the algorithm; B is the cost to the adversary; and g is the number of good jobs in the input. Then:*

$$E\left(A - \sqrt{Bg\gamma}\right) \geq 0$$

Proof. By Lemma 16 the estimator has γ estimation gap, and returns the exact same values whether the number of actual good jobs is g_1 or g_2. Thus, the server can set the price based only on the job index. So, for each index $i \in [1, n]$, let c_i be the price for the i-th job; and let $C = \sum_{i=1}^{n} c_i$. Let $E_1(A)$ (resp. $E_1(B)$) be the expected cost to the algorithm (resp. the expected adversarial total cost) for distribution 1; and $E_2(A)$, $E_2(B)$ be analogous expected costs for distribution 2.

To show $E(A - \sqrt{Bg\gamma}) \geq 0$, we will show that $E(A)/\sqrt{E(B)g} \geq \sqrt{\gamma}$, and then cross-multiply and use Jensen's inequality. If we define a random variable a_i that has value c_i if the i-th job is good and 0 otherwise, and note that $A = \sum_{i=1}^{n} a_i$, then by linearity of expectation, the expected algorithm

cost due to fees in distribution 1 is $\sum_{i=1}^{n} E(a_i) = C\gamma g_0/n$, with service cost of n. A similar analysis for $E_1(B)$ gives:

$$E_1(A) = C\gamma g_0/n + n$$
$$E_1(B) = \frac{n - \gamma g_0}{n} C.$$

Similarly, for the second distribution:

$$E_2(A) = Cg_0/(\gamma n) + n$$
$$E_2(B) = \frac{n - g_0/\gamma}{n} C.$$

Hence, we have:

$$\begin{aligned}\frac{E_1(A)}{\sqrt{E_1(B)g_1}} &= \sqrt{\frac{C\gamma g_0}{n(n-\gamma g_0)}} + \sqrt{\frac{n^3}{C\gamma g_0(n-\gamma g_0)}} \\ &= \frac{V_1 + n/V_1}{\sqrt{(n-\gamma g_0)}}\end{aligned}$$

where $V_1 = \sqrt{\frac{C\gamma g_0}{n}}$. Also:

$$\begin{aligned}\frac{E_2(A)}{\sqrt{E_2(B)g_2}} &= \sqrt{\frac{C(g_0/\gamma)}{n(n-g_0/\gamma)}} + \sqrt{\frac{n^3}{C(g_0/\gamma)(n-g_0/\gamma)}} \\ &= \frac{V_2 + n/V_2}{\sqrt{(n-g_0/\gamma)}}\end{aligned}$$

where $V_2 = \sqrt{\frac{Cg_0/\gamma}{n}} = V_1/\gamma$.

Let $f(C)$ be the expected value of the ratio of $E(A)/\sqrt{E(B)g}$ as a function of C. Since the adversary chooses each distribution with probability $1/2$, we have:

$$\begin{aligned}f(C) &= 1/2 \left(\frac{V_1 + n/V_1}{\sqrt{n-\gamma g_0}} + \frac{V_1/\gamma + n\gamma/V_1}{\sqrt{n - g_0/\gamma}} \right) \\ &\geq \frac{1}{2(\sqrt{n-\gamma/g_0})} \left((1 + 1/\gamma)V_1 + (1+\gamma)n/V_1 \right))\end{aligned}$$

Consider the expression in parenthesis above: $g(V_1) = (1+1/\gamma)V_1 + (1+\gamma)n/V_1$ The derivative of $g(V_1)$ with respect to V_1 is $1/\gamma + 1 - (\gamma+1)n/V_1^2$. From this, we see that $g(V_1)$ is minimized when $V_1 = \sqrt{n\gamma}$. Plugging this back in to f, we get that $f(C) \geq \sqrt{\gamma}$.

$$\begin{aligned} f(C) &\geq \frac{1}{2(\sqrt{n - \gamma/g_0})} \left((1+1/\gamma)V_1 + (1+\gamma)(n/V_1)\right) \\ &= \frac{1}{2(\sqrt{n - \gamma/g_0})} \left(\sqrt{n\gamma} + \sqrt{n/\gamma} + \sqrt{n/\gamma} + \sqrt{n\gamma}\right) \end{aligned}$$

$$\begin{aligned} &= \frac{1}{2(\sqrt{n - \gamma/g_0})} \left(2\sqrt{n\gamma} + 2\sqrt{n/\gamma}\right) \\ &= \frac{\sqrt{n\gamma} + \sqrt{n/\gamma}}{\sqrt{n - \gamma/g_0}} \\ &\geq \frac{\sqrt{n\gamma} + \sqrt{n/\gamma}}{\sqrt{n}} \\ &\geq \sqrt{\gamma} \end{aligned}$$

Hence, we have proven that the expected value of the ratio $E(A)/\sqrt{E(B)g\gamma}$ is always at least 1, for our adversarial input distribution, for any deterministic algorithm. Cross multiplying, we get

$$\begin{aligned} E(A) &\geq \sqrt{E(B)g\gamma} \\ &\geq E(\sqrt{Bg\gamma}) \end{aligned}$$

where the second line holds by applying Jensen's inequality [86] to the convex function[1] $f(x) = -\sqrt{x}$. Finally, by linearity of expectation, this inequality implies that $E(A - \sqrt{Bg\gamma}) \geq 0$, which completes the argument. □

THEOREM 3. *Let γ be any positive integer; g_0 be any positive multiple of γ; and n any multiple of γg_0. Next, fix any randomized algorithm in our model. Then, there is an input with n jobs, g of which are good for $g \in \{\gamma g_0, g_0/\gamma\}$;*

[1] Equivalently, we can directly apply the (less well-known) version of Jensen's inequality for concave functions [42] to obtain the second line.

and an estimator with estimation gap γ on that input. Additionally, the randomized algorithm on that input has expected cost:

$$E(A) = \Omega\left(E(\sqrt{\gamma B(g+1)}) + \gamma(g+1)\right).$$

Proof. By Lemma 17, on our adversarial distribution, any deterministic algorithm has $E(A) - \sqrt{\gamma E(B) g\gamma} \geq 0$.

We fix values of g_0, n and γ in our adversarial distribution and define a game theoretic payoff matrix $\mathcal{M}$ as follows. The rows of $\mathcal{M}$ correspond to deterministic inputs in the sample space used by our adversarial distribution; the columns correspond to all deterministic algorithms over n jobs. Further, for each input i, and algorithm j, the matrix entry $\mathcal{M}(i,j)$ is the value $A - \sqrt{Bg\gamma}$, where A and B are the costs to the algorithm and adversary respectively, when algorithm j is run on input i; and the value g is the number of good jobs in input i.

Lemma 17 proves that the minimax of this payoff matrix is at least 0. So by Von Neumann's minimax theorem [94, 123], the maximin is also at least 0. In other words, via Yao's principal [123], any *randomized* algorithm has some *deterministic* input for which $E(A - \sqrt{\gamma Bg}) \geq 0$. By linearity of expectation:

$$E(A) \geq E(\sqrt{\gamma Bg}).$$

Finally, every algorithm has job service cost that equals $n = \gamma g$, so $E(A) = n \geq \gamma g$. Adding this to the above inequality gives that $2E(A) \geq E(\sqrt{\gamma Bg}) + \gamma g$, or $E(A) = \Omega(\sqrt{\gamma E(B) g} + \gamma g)$. Since g is positive, this value is:

$$\Omega\left(\sqrt{\gamma E(B)(g+1)} + \gamma(g+1)\right).$$

Since the sample space of our adversarial distribution has $g \in \{\gamma g_0, g_0/\gamma\}$, the worst-case distribution chosen for the maximin will also have $g \in \{\gamma g_0, g_0/\gamma\}$. □

9. Preliminary Simulation Results

We conduct simulations for input distributions where there is a large attack and all good jobs are serviced as late as possible; here, our aim is to simply verify the scaling behavior predicted by theory. Our custom simulations are written in Python and the source code is available to the public via GitHub [23].

m	$g(I_m)$	$\hat{g}(I_m)$
1	bad	1
2	bad	1
3	bad	0
4	bad	0
5	bad	0
6	bad	0
7	good	0
8	good	1

m	$g(I_m)$	$\hat{g}(I_m)$
1	bad	1
2	bad	1
3	bad	1
4	bad	1
5	good	0
6	good	0
7	good	0
8	good	1

Figure 6: Two small input examples for experiments with LINEAR for $n = 8$, with $\gamma = 2$ (left) and $\gamma = 4$ (right). Red dashed lines indicate the end of an iteration.

9.1. Experiments for LINEAR

In our experiments for LINEAR, the distribution of good and bad jobs, along with the performance of the estimator, is set up as follows. Let the jobs be indexed as $J_1, J_2, \ldots, J_n$. Then, let $\mathcal{I}_m$ be the time interval containing the single point in time at which the m-th job is generated for $1 \leq m \leq n$.

- For $m \in [1, \gamma]$, job J_m is bad and $\hat{g}(\mathcal{I}_m) = 1$
- For $m \in [\gamma + 1, n - \gamma]$, job J_m is bad and $\hat{g}(\mathcal{I}_m) = 0$
- For $m \in [n - \gamma + 1, n - 1]$, job J_m is good and $\hat{g}(\mathcal{I}_m) = 0$
- For $m = n$, job J_m is good and $\hat{g}(\mathcal{I}_m) = 1$

For all other intervals $\mathcal{I}_t$ consisting of single points in time, t, $\hat{g}(\mathcal{I}_t) = 0$. Then, the additive property defines the estimator output for all larger intervals.

Thus, the first γ jobs are bad, but considered to be good by the estimator (each such job will end an iteration); the next $n - 2\gamma$ jobs are bad and considered to be bad by the estimator; the next $\gamma - 1$ jobs are good and considered to be bad by the estimator; and the last job is good and considered to be good by the estimator (which ends an iteration).

We highlight that, in the above setup, the estimator errs in a way that favors the adversary and disfavors our algorithm. Given that the first γ bad

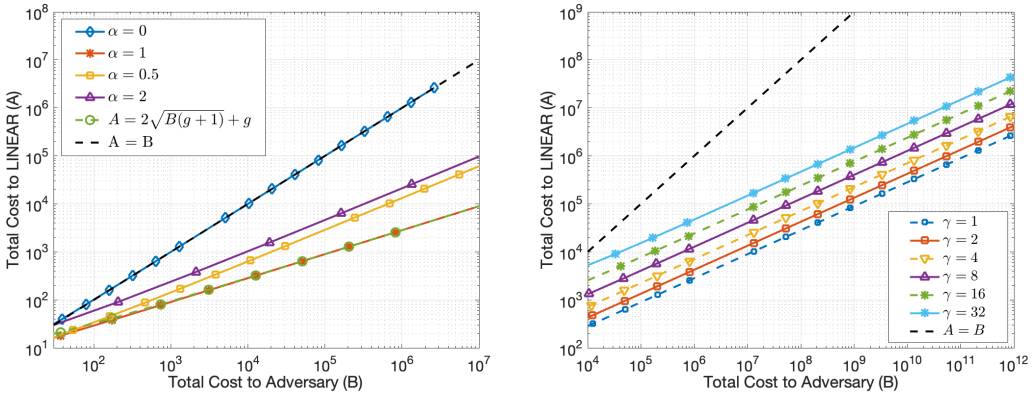

Figure 7: Results for LINEAR. (Left) $\gamma = 1$; α value varies. (Right) $\alpha = 1$; γ value varies.

jobs are (erroneously) considered good by the estimator, the adversary pays only 1 for each such job, since each such job ends an iteration. Thus, as γ grows, the adversary benefits; see the examples for $\gamma = 2$ and $\gamma = 4$ in Figure 6. Additionally, of the final γ (good) jobs, all but the last one are correctly identified as good, which increases the cost to the algorithm.

Specific Experiments. We conduct two experiments. For our first experiment, we let $n = 10 \times 2^x$ for integer $x \in [-1, 18]$. Then, we fix $\gamma = 1$ and investigate the impact of different values of α (recall Section 6); specifically, we let $\alpha = 0, 1/2, 1$ and 2.

For our second experiment, we set $n = 10 \times 2^x$, where $x \in [3, 18]$. We fix $\alpha = 1$ and investigate the impact of different values of γ; specifically, we let $\gamma = 1, 2, 4, 8, 16$ and 32. The range for x is slightly smaller than in the first experiments, since the smallest power-of-2-sized input that holds both good and bad jobs is $10 \times 2^3 = 80$ when $\gamma = 32$.

Results for LINEAR. The results of our first experiment are plotted in Figure 7 (Left). The algorithm's cost is denoted by A, while the adversary's cost is denoted by B. The results show scaling consistent with our theoretical analysis. Notably, setting $\alpha = 1$ achieves the most advantageous cost for algorithm versus the adversary. We observe that for $\alpha = 0, 2$ and 0.5, the algorithm cost increases by up to a factor of approximately 33.3, 3.3, and 1.8, respectively, relative to LINEAR (i.e., $\alpha = 1$) at $B = 10^4$. We include the line $A = B$ as a reference, and this closely corresponds to the case for $\alpha = 0$ (i.e., each job has cost 1), since the service cost incurred by the algorithm is

close to the number of bad jobs in these experiments. We also include the equation $A = 2(\sqrt{B(g+1)} + g)$, which closely fits the line for $\alpha = 1$.

The results of our second experiment are plotted in Figure 7 (Right). As expected, when γ increases—and, thus, the accuracy of our estimator worsens—the cost ratio of the algorithm to the adversary increases. Specifically, for $B = 10^7$, we see that the values of $\gamma = 2, 4, 8, 16$ and 32 correspond to a cost ratio that is, respectively, a factor of 1.5, 2.6, 4.4, 8.8, and 15.5 larger than than for $\gamma = 1$. Given this, and noting the fairly even spacing of the trend lines on the log-log-scaled plot, the cost appears to scale according to a (small) polynomial in γ. We note that, despite this behavior, the cost of the algorithm versus the adversary is still below the $A = B$ line; thus, LINEAR still achieves a significant advantage for the values of γ tested.

9.2. Experiments for LINEAR-POWER

To evaluate a challenging case for LINEAR-POWER, we create an input causing many bounced good jobs in the first iteration. From the start of this iteration, we consider disjoint periods of 2Δ seconds. Each period $i \geq 0$ will have 2^i bad jobs serviced, followed by $(i+1)M$ good jobs that are bounced. For example, in the first period, we have 1 bad job serviced followed by M good jobs; these M good jobs will be bounced, since they start with a solution to a 1-hard challenge, while the current price has increased to 2 due to the first bad job. In the second period, two bad jobs are serviced, followed by M new good jobs, each attaching a fee of 1. These are followed by the M good jobs that were bounced in the first period. All of these $2M$ good jobs will be bounced, since the current price has increased to 4 due to the two bad jobs.

For each integer $z \in [10, 28]$, we fix a number of bad jobs $b = 2^z - 1$, and let M range over values $1, 2, 4$, and 8. Once all b bad jobs have been serviced — which occurs in period z — a final batch of M good jobs arrive with a 1 hard solution and these are bounced. However, the other $(z+1)M$ (good) jobs will start being serviced. Specifically, γ of these (good) jobs will be serviced, with the first $\gamma - 1$ jobs (erroneously) considered bad by the estimator, and γ-th job (correctly) considered good; this ends the first iteration. All remaining good jobs will be serviced in subsequent iterations in similar fashion (γ per iteration), since the current price never again exceeds $2^{\lfloor \log(\gamma(\gamma+1)) \rfloor} \leq \gamma(\gamma+1)$ (recall Lemma 2).

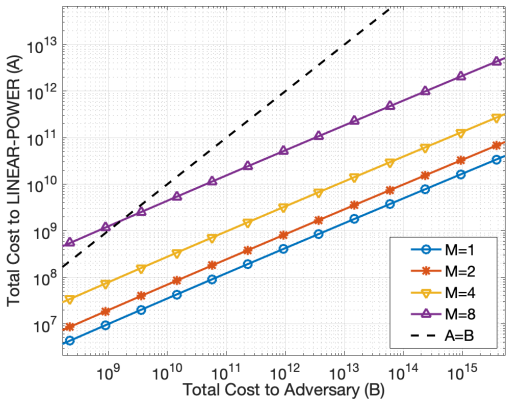

Figure 8: Our preliminary simulation results for LINEAR-POWER. Different values of M are evaluated with $\gamma = 8$.

Results for LINEAR-POWER. Figure 8 illustrates the results of our simulations when $\gamma = 8$. As predicted by our upper bound on cost in Theorem 2, LINEAR-POWER achieves a lower cost than the adversary for B sufficiently large. However, aligning with our analysis, we see that this cost advantage decreases as M grows. In particular, between $M = 1$ and $M = 8$, we see that the cost ratio decreases from roughly 2×10^3 to 20 for a fixed $B = 9 \times 10^{11}$. Very similar results are observed for other values of γ, and so we omit these and note that, for this set of experiments, the algorithm's cost appears to be primarily sensitive to M.

10. Conclusion and Future Work

We have provided two deterministic algorithms for DoS defense. Our algorithms dynamically adjust the server's prices based on feedback from an estimator which estimates the number of good jobs in any previously observed time interval. The accuracy of the estimator on the input, the number and temporal placement of the good and bad jobs, and the adversary's total spending are all unknown to the algorithm, but these all affect the algorithm's cost.

Critically, during a significant attack the cost to our algorithm is *asymptotically* less than the cost to the adversary. We believe this is a important property for deterring DoS attacks. We have also given a lower bound for

randomized algorithms, showing that our algorithmic costs are asymptotically tight, whenever γ is a constant. We note that, while our algorithms are deterministic, our lower bound holds even for randomized algorithms.

Many interesting problems remain. Are there less restrictive properties for the estimator which suffice to achieve a good pricing strategy? Can we model both the clients and the server as selfish but rational agents? In particular, can we design a mechanism that results in a Nash equilibrium, and also provides good cost performance as a function of the adversarial cost?

Acknowledgements. This material is based upon work supported by the National Science Foundation under grant numbers CNS-2210299, CNS-2210300, and CCF-2144410.

References

[1] Martin Abadi, Mike Burrows, Mark Manasse, and Ted Wobber. Moderately hard, memory-bound functions. *ACM Transactions on Internet Technology*, 5(2):299–327, 2005.

[2] Abhinav Aggarwal, Varsha Dani, Thomas Hayes, and Jared Saia. Secure One-Way Interactive Communication. In *Proceedings of the 15th International Conference on Distributed Computing and Networking (ICDCN)*, 2017.

[3] Isra Mohamed Ali, Maurantonio Caprolu, and Roberto Di Pietro. Foundations, properties, and security applications of puzzles: A survey. *ACM Computing Survey*, 53(4):1–38, 2020.

[4] Wael Alosaimi, Michal Zak, and Khalid Al-Begain. Denial of service attacks mitigation in the cloud. In *2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies*, pages 47–53. IEEE, 2015.

[5] Lakshmi Anantharamu and Bogdan S Chlebus. Broadcasting in ad hoc multiple access channels. *Theoretical Computer Science*, 584:155–176, 2015.

[6] Lakshmi Anantharamu, Bogdan S Chlebus, Dariusz R Kowalski, and Mariusz A Rokicki. Deterministic broadcast on multiple access channels. In *2010 Proceedings IEEE INFOCOM*, pages 1–5. IEEE, 2010.

[7] Tuomas Aura, Pekka Nikander, and Jussipekka Leiwo. DoS-resistant authentication with client puzzles. In *International Workshop on Security Protocols*, pages 170–177. Springer, 2000.

[8] Rudolf Avenhaus, Bernhard Von Stengel, and Shmuel Zamir. Inspection games. *Handbook of game theory with economic applications*, 3:1947–1987, 2002.

[9] AWS. AWS best practices for DDoS resiliency, 2019.

[10] KRWV Bandara, T Abeysinghe, A Hijaz, DGT Darshana, H Aneez, SJ Kaluarachchi, KD Sulochana, and Mr DhishanDhammearatchi. Preventing ddos attack using data mining algorithms. *International Journal of Scientific and Research Publications*, 6(10):390, 2016.

[11] Mercedes Barrionuevo, Mariela Lopresti, Natalia Miranda, and Fabiana Piccoli. An anomaly detection model in a lan using k-nn and high performance computing techniques. In *Computer Science–CACIC 2017: 23rd Argentine Congress, La Plata, Argentina, October 9-13, 2017, Revised Selected Papers 23*, pages 219–228. Springer, 2018.

[12] Michael A. Bender, Jeremy T. Fineman, Seth Gilbert, John Kuszmaul, and Maxwell Young. Fully energy-efficient randomized backoff: Slow feedback loops yield fast contention resolution. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing (PODC)*, page 231–242, 2024.

[13] Michael A. Bender, Jeremy T. Fineman, Seth Gilbert, and Maxwell Young. How to Scale Exponential Backoff: Constant Throughput, Polylog Access Attempts, and Robustness. In *Proceedings of the* 27^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 636–654, 2016.

[14] Michael A. Bender, Jeremy T. Fineman, Mahnush Movahedi, Jared Saia, Varsha Dani, Seth Gilbert, Seth Pettie, and Maxwell Young. Resource-competitive algorithms. *SIGACT News*, 46(3):57–71, September 2015.

[15] Nikita Borisov. Computational puzzles as Sybil defenses. In 6^{th} *IEEE International Conference on Peer-to-Peer Computing (P2P)*, pages 171–176, 2006.

[16] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. cambridge university press, 2005.

[17] Allan Borodin, Jon Kleinberg, Prabhakar Raghavan, Madhu Sudan, and David P Williamson. Adversarial queuing theory. *Journal of the ACM (JACM)*, 48(1):13–38, 2001.

[18] Rodrigo Braga, Edjard Mota, and Alexandre Passito. Lightweight DDoS flooding attack detection using NOX/OpenFlow. In *IEEE local computer network conference*, pages 408–415. IEEE, 2010.

[19] Chaitanya Buragohain and Nabajyoti Medhi. Flowtrapp: An sdn based architecture for ddos attack detection and mitigation in data centers. In *2016 3rd International Conference on Signal Processing and Integrated Networks (SPIN)*, pages 519–524. IEEE, 2016.

[20] Jin Cao, William S Cleveland, Yuan Gao, Kevin Jeffay, F Donelson Smith, and Michele Weigle. Stochastic models for generating synthetic http source traffic. In *IEEE INFOCOM 2004*, volume 3, pages 1546–1557. IEEE, 2004.

[21] Jin Cao, William S Cleveland, Dong Lin, and Don X Sun. Internet traffic tends toward poisson and independent as the load increases. *Nonlinear estimation and classification*, pages 83–109, 2003.

[22] Glenn Carl, George Kesidis, Richard R Brooks, and Suresh Rai. Denial-of-service attack-detection techniques. *IEEE Internet computing*, 10(1):82–89, 2006.

[23] Trisha Chakraborty. Simulation Code. https://github.com/trishac97/RB-Cost-DDOS, 2022.

[24] Trisha Chakraborty, Shaswata Mitra, and Sudip Mittal. CAPoW: Context-aware ai-assisted proof of work based ddos defense. In *Proceedings of the 20th International Conference on Security and Cryptography (SECRYPT)*, pages 62–72, 2023.

[25] Trisha Chakraborty, Jared Saia, and Maxwell Young. Defending Hash Tables from Subterfuge with Depth Charge. In *Proceedings of the 25th International Conference on Distributed Computing and Networking*

(ICDCN), page 134–143, New York, NY, USA, 2024. Association for Computing Machinery.

[26] Nathanael Chambers, Ben Fry, and James McMasters. Detecting denial-of-service attacks from social media text: Applying nlp to computer security. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1626–1635, 2018.

[27] Haimin Chen and Chaodong Zheng. Broadcasting competitively against adaptive adversary in multi-channel radio networks. In *24th International Conference on Principles of Distributed Systems, OPODIS*, volume 184, pages 22:1–22:16, 2020.

[28] Yu Chen, Wei-Shinn Ku, Kazuya Sakai, and Christopher DeCruze. A novel DDoS attack defending framework with minimized bilateral damages. In *2010 7th IEEE Consumer Communications and Networking Conference*, pages 1–5. IEEE, 2010.

[29] Chen-Mou Cheng, HT Kung, and Koan-Sin Tan. Use of spectral analysis in defense against dos attacks. In *Global Telecommunications Conference, 2002. GLOBECOM'02. IEEE*, volume 3, pages 2143–2148. IEEE, 2002.

[30] Chi Cheng, Wee Peng Tay, and Guang-Bin Huang. Extreme learning machines for intrusion detection. In *The 2012 International joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2012.

[31] Bogdan S Chlebus, Dariusz R Kowalski, and Mariusz A Rokicki. Maximum throughput of multiple access channels in adversarial environments. *Distributed Computing*, 22(2):93–116, 2009.

[32] Hyoung-Kee Choi and John O Limb. A behavioral model of web traffic. In *Proceedings. Seventh International Conference on Network Protocols*, pages 327–334. IEEE, 1999.

[33] Vikas Chouhan and Sateesh Kumar Peddoju. Hierarchical storage technique for maintaining hop-count to prevent ddos attack in cloud computing. In *Proceedings of International Conference on Advances in Computing*, pages 511–518. Springer, 2012.

[34] Allen Clement, Harry Li, Jeff Napper, Jean-Philippe Martin, Lorenzo Alvisi, and Mike Dahlin. Bar primer. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, pages 287–296. IEEE, 2008.

[35] Allen Clement, Jeff Napper, Harry Li, Jean-Philipe Martin, Lorenzo Alvisi, and Michael Dahlin. Theory of bar games. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 358–359, 2007.

[36] R.L. Cruz. A calculus for network delay. i. network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, 1991.

[37] Alberto Dainotti, Antonio Pescapé, and Giorgio Ventre. A cascade architecture for dos attacks detection based on the wavelet transform. *Journal of Computer Security*, 17(6):945–968, 2009.

[38] Varsha Dani, Mahnush Movahedi, Jared Saia, and Maxwell Young. Interactive Communication with Unknown Noise Rate. In *Proceedings of the Colloquium on Automata, Languages, and Programming (ICALP)*, 2015.

[39] C Dingankar and RR Brooks. Denial of service games. In *Proceedings of the third annual cyber security and information infrastructure research workshop*, pages 7–17. Citeseer, 2007.

[40] Colin Dixon, Thomas E Anderson, and Arvind Krishnamurthy. Phalanx: Withstanding multimillion-node botnets. In *NSDI*, volume 8, pages 45–58, 2008.

[41] Roberto Doriguzzi-Corin, Stuart Millar, Sandra Scott-Hayward, Jesus Martinez-del Rincon, and Domenico Siracusa. LUCID: A practical, lightweight deep learning solution for DDoS attack detection. *IEEE Transactions on Network and Service Management*, 17(2):876–889, 2020.

[42] Sever Silvestru Dragomir. Some refinements of Jensen's inequality. *J. Math. Anal. Appl*, 168(2):518–522, 1992.

[43] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Annual international cryptology conference*, pages 139–147. Springer, 1992.

[44] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Proceedings of the* 12^{th} *Annual International Cryptology Conference on Advances in Cryptology*, pages 139–147, 1993.

[45] Mehran Fallah. A puzzle-based defense strategy against flooding attacks using game theory. *IEEE transactions on dependable and secure computing*, 7(1):5–19, 2008.

[46] Fei Fang, Thanh H Nguyen, Rob Pickles, Wai Y Lam, Gopalasamy R Clements, Bo An, Amandeep Singh, Brian C Schwedock, Milin Tambe, and Andrew Lemieux. Paws—a deployed game-theoretic application to combat poaching. *AI Magazine*, 38(1):23–36, 2017.

[47] Fei Fang, Peter Stone, and Milind Tambe. When security games go green: Designing defender strategies to prevent poaching and illegal fishing. In *IJCAI*, pages 2589–2595, 2015.

[48] Wu-chang Feng and Ed Kaiser. kapow webmail: Effective disincentives against spam. *Proc. of 7th CEAS*, 2010.

[49] Pawel Garncarek, Tomasz Jurdzinski, and Dariusz R. Kowalski. Local queuing under contention. In *32nd International Symposium on Distributed Computing (DISC)*, volume 121, pages 28:1–28:18, 2018.

[50] Paweł Garncarek, Tomasz Jurdziński, and Dariusz R. Kowalski. Stable Memoryless Queuing under Contention. In *33rd International Symposium on Distributed Computing (DISC)*, volume 146, pages 17:1–17:16, 2019.

[51] Seth Gilbert, Valerie King, Seth Pettie, Ely Porat, Jared Saia, and Maxwell Young. (Near) optimal resource-competitive broadcast with jamming. In *Proceedings of the* 26^{th} *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 257–266, 2014.

[52] Seth Gilbert, Calvin Newport, Nitin H. Vaidya, and Alex Weaver. Contention resolution with predictions. In *PODC '21: ACM Symposium on Principles of Distributed Computing*, pages 127–137. ACM, 2021.

[53] Seth Gilbert and Maxwell Young. Making Evildoers Pay: Resource-Competitive Broadcast in Sensor Networks. In *Proceedings of the* 31^{th} *Symposium on Principles of Distributed Computing (PODC)*, pages 145–154, 2012.

[54] Seth Gilbert and Chaodong Zheng. Sybilcast: Broadcast on the open airwaves. In *Proceedings of the* 25^{th} *Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 130–139, 2013.

[55] Jesus M Gonzalez, Mohd Anwar, and James BD Joshi. A trust-based approach against ip-spoofing attacks. In *2011 Ninth Annual International Conference on Privacy, Security and Trust*, pages 63–70. IEEE, 2011.

[56] Diksha Gupta, Jared Saia, and Maxwell Young. Proof of work without all the work. In *Proceedings of the* 19^{th} *International Conference on Distributed Computing and Networking (ICDCN)*, 2018.

[57] Diksha Gupta, Jared Saia, and Maxwell Young. Resource burning for permissionless systems. In *International Colloquium on Structural Information and Communication Complexity*, pages 19–44. Springer, 2020.

[58] Diksha Gupta, Jared Saia, and Maxwell Young. Bankrupting Sybil despite churn. In *Proceedings of the 41st IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2021.

[59] Diksha Gupta, Jared Saia, and Maxwell Young. Bankrupting sybil despite churn. *Journal of Computer and System Sciences*, 135:89–124, 2023.

[60] Qiang He, Cheng Wang, Guangming Cui, Bo Li, Rui Zhou, Qingguo Zhou, Yang Xiang, Hai Jin, and Yun Yang. A game-theoretical approach for mitigating edge DDoS attack. *IEEE Transactions on Dependable and Secure Computing*, 19(4):2333–2348, 2022.

[61] Radware Inc. DDoS Attacks History. https://www.radware.com/security/ddos-knowledge-center/ddos-chronicles/ddos-attacks-history, 2017.

[62] Ari Juels and John Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, pages 151–165, 1999.

[63] Ed Kaiser and Wu chang Feng. mod_kapow: Mitigating dos with transparent proof-of-work. In *Proceedings of the 2007 ACM CoNEXT conference*, pages 1–2, 2007.

[64] Srikanth Kandula, Dina Katabi, Matthias Jacob, and Arthur Berger. Botz-4-sale: Surviving organized ddos attacks that mimic flash crowds. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 287–300, 2005.

[65] Vijay Katkar, Amol Zinjade, Suyed Dalvi, Tejal Bafna, and Rashmi Mahajan. Detection of DoS/DDoS attack against HTTP servers using naive bayesian. In *2015 International Conference on Computing Communication Control and Automation*, pages 280–285. IEEE, 2015.

[66] Bashar Ahmed Khalaf, Salama A Mostafa, Aida Mustapha, Mazin Abed Mohammed, and Wafaa Mustafa Abduallah. Comprehensive review of artificial intelligence and statistical approaches in distributed denial of service attack and defense methods. *IEEE Access*, 7:51691–51713, 2019.

[67] Soon Hin Khor and Akihiro Nakao. spow: On-demand cloud-based eddos mitigation mechanism. In *HotDep (Fifth Workshop on Hot Topics in System Dependability)*, 2009.

[68] Yoohwan Kim, Wing Cheong Lau, Mooi Choo Chuah, and H Jonathan Chao. Packetscore: Statistics-based overload control against distributed denial-of-service attacks. In *IEEE INFOCOM 2004*, volume 4, pages 2594–2604. IEEE, 2004.

[69] Yoohwan Kim, Wing Cheong Lau, Mooi Choo Chuah, and H Jonathan Chao. Packetscore: a statistics-based packet filtering scheme against distributed denial-of-service attacks. *IEEE transactions on dependable and secure computing*, 3(2):141–155, 2006.

[70] Valerie King, Jared Saia, and Maxwell Young. Conflict on a Communication Channel. In *Proceedings of the* 30^{th} *Symposium on Principles of Distributed Computing (PODC)*, pages 277–286, 2011.

[71] Jon Kleinberg and Eva Tardos. *Algorithm design (see Chapter 4)*. Pearson Education India, 2006.

[72] Madarapu Naresh Kumar, Raju Korra, P Sujatha, and Mu Kumar. Mitigation of economic distributed denial of sustainability (eddos) in cloud computing. In *Proc. of the Intl'Conf. on Advances in Engineering and Technology*, 2011.

[73] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. *ACM SIGCOMM computer communication review*, 34(4):219–230, 2004.

[74] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In 14-*th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1859–1877, 2020.

[75] Hyeonmin Lee, Taehyun Kang, Sukhun Yang, Jinyong Jun, and Taekyoung Kwon. Ddd: A dns-based ddos defense scheme using puzzles. In *Proceedings of the 33rd International Conference on Computer Communications and Networks (ICCCN)*, pages 1–9, 2024.

[76] Frank Li, Prateek Mittal, Matthew Caesar, and Nikita Borisov. SybilControl: Practical Sybil defense with computational puzzles. In *Proceedings of the Seventh ACM Workshop on Scalable Trusted Computing*, pages 67–78, 2012.

[77] Muhai Li and Ming Li. A new approach for detecting ddos attacks based on wavelet analysis. In *2009 2nd International Congress on Image and Signal Processing*, pages 1–5. IEEE, 2009.

[78] Qing Li, He Huang, Ruoyu Li, Jianhui Lv, Zhenhui Yuan, Lianbo Ma, Yi Han, and Yong Jiang. A comprehensive survey on ddos defense systems: New trends and challenges. *Computer Networks*, page 109895, 2023.

[79] Yang Li, Li Guo, Zhi-Hong Tian, and Tian-Bo Lu. A lightweight web server anomaly detection method based on transductive scheme and genetic algorithms. *Computer Communications*, 31(17):4018–4025, 2008.

[80] Iuon-Chang Lin and Tzu-Chun Liao. A survey of blockchain security issues and challenges. *IJ Network Security*, 19(5):653–659, 2017.

[81] Debin Liu and L. Jean Camp. Proof of work can work. In *Proceedings of the 5^{th} Workshop on the Economics of Information Security (WEIS)*, 2006.

[82] Zaoxing Liu, Hun Namkung, Georgios Nikolaidis, Jeongkeun Lee, Changhoon Kim, Xin Jin, Vladimir Braverman, Minlan Yu, and Vyas Sekar. Jaqen: A {High-Performance}{Switch-Native} approach for detecting and mitigating volumetric {DDoS} attacks with programmable switches. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3829–3846, 2021.

[83] Zhen Liu, Nicolas Niclausse, and César Jalpa-Villanueva. Traffic model and performance evaluation of web servers. *Performance Evaluation*, 46(2-3):77–100, 2001.

[84] Andrey Lukyanenko, Vladimir Mazalov, Andrei Gurtov, and Igor Falko. Playing defense by offense: Equilibrium in the dos-attack problem. In *The IEEE symposium on Computers and Communications*, pages 433–436. IEEE, 2010.

[85] Nancy A Lynch. *Distributed algorithms*. Elsevier, 1996.

[86] Edward James McShane. Jensen's inequality. *Bulletin of the American Mathematical Society*, 43(8):521–527, 1937.

[87] Michael Mitzenmacher. A model for learned bloom filters, and optimizing by sandwiching. *CoRR*, abs/1901.00902, 2019.

[88] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge University Press, 2017.

[89] Michael Mitzenmacher and Sergei Vassilvitskii. *Algorithms with Predictions. In Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press, 2021.

[90] Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *Communications of the ACM*, 65 (7):33–35, July 2022.

[91] Jad Naous, David Erickson, G Adam Covington, Guido Appenzeller, and Nick McKeown. Implementing an openflow switch on the netfpga platform. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 1–9, 2008.

[92] Jema David Ndibwile, A Govardhan, Kazuya Okada, and Youki Kadobayashi. Web server protection against application layer DDoS attacks using machine learning and traffic authentication. In *Proceedings of the 39th IEEE Annual Computer Software and Applications Conference*, pages 261–267, 2015.

[93] Hoai-Vu Nguyen and Yongsun Choi. Proactive detection of ddos attacks utilizing k-nn classifier in an anti-ddos framework. *International Journal of Computer and Information Engineering*, 4(3):537–542, 2010.

[94] Hukukane Nikaidô et al. On von Neumann's minimax theorem. *Pacific J. Math*, 4(1):65–72, 1954.

[95] M. A. Noureddine, A. M. Fawaz, A. Hsu, C. Guldner, S. Vijay, T. Başar, and W. H. Sanders. Revisiting client puzzles for state exhaustion attacks resilience. In *Proceedings of the 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 617–629, 2019.

[96] Georgios Oikonomou and Jelena Mirkovic. Modeling human behavior for defense against flash-crowd attacks. In *Proceedings of the IEEE International Conference on Communications*, pages 1–6. IEEE, 2009.

[97] Bryan Parno, Dan Wendlandt, Elaine Shi, Adrian Perrig, Bruce Maggs, and Yih-Chun Hu. Portcullis: Protecting connection setup from denial-of-capability attacks. *ACM SIGCOMM Computer Communication Review*, 37(4):289–300, 2007.

[98] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman

Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 9684–9693, 2018.

[99] Ziv Scully, Isaac Grosof, and Michael Mitzenmacher. Uniform bounds for scheduling with job size estimates. In *13th Innovations in Theoretical Computer Science Conference ITCS*, volume 215, pages 114:1–114:30, 2022.

[100] Elaine Shi, Ion Stoica, David Andersen, and Adrian Perrig. Overdose: A generic ddos protection service using an overlay network. Technical report, Technical Report CMU-CS-06-114, Carnegie Mellon University, 2006.

[101] Gaurav Somani, Manoj Singh Gaur, Dheeraj Sanghi, Mauro Conti, Muttukrishnan Rajarajan, and Rajkumar Buyya. Combating ddos attacks in the cloud: requirements, trends, and future directions. *IEEE Cloud Computing*, 4(1):22–32, 2017.

[102] T. Spyridopoulos, G. Karanikas, T. Tryfonas, and G. Oikonomou. A game theoretic defence framework against dos/ddos cyber attacks. *Computers & Security*, 38:39–50, 2013.

[103] Milind Tambe. *Security and game theory: algorithms, deployed systems, lessons learned*. Cambridge university press, 2011.

[104] Zhiyuan Tan, Aruna Jamdagni, Xiangjian He, Priyadarsi Nanda, and Ren Ping Liu. A system for denial-of-service attack detection based on multivariate correlation analysis. *IEEE transactions on parallel and distributed systems*, 25(2):447–456, 2013.

[105] TechHQ Technology and business. DDoS attacks cost US businesses 10bn per year. https://techhq.com/2019/03/ddos-attacks-cost-us-businesses-10bn-per-year, 2019.

[106] Reza Tourani, George Torres, and Satyajayant Misra. Persia: A puzzle-based interest flooding attack countermeasure. In *Proceedings of the 7th ACM Conference on Information-Centric Networking*, pages 117–128, 2020.

[107] Chang-Lung Tsai, Allen Y Chang, and Ming-Szu Huang. Early warning system for ddos attacking based on multilayer deployment of time delay neural network. In *2010 Sixth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pages 704–707. IEEE, 2010.

[108] James Victor Uspensky. *Introduction to mathematical probability*. McCraw-hill, 1937.

[109] Rajagopalan Vijayasarathy, Serugudi Venkataraman Raghavan, and Balaraman Ravindran. A system approach to network modeling for ddos detection using a naive bayesian classifier. In *2011 Third International Conference on Communication Systems and Networks (COMSNETS 2011)*, pages 1–10. IEEE, 2011.

[110] Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford. CAPTCHA: Using hard AI problems for security. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*, pages 294–311. Springer, 2003.

[111] Luis Von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895):1465–1468, 2008.

[112] Bernhard von Stengel. *Recursive inspection games*. Univ. der Bundeswehr München, Fak. für Informatik, 1991.

[113] Michael Walfish, Mythili Vutukuru, Hari Balakrishnan, David Karger, and Scott Shenker. DDoS defense by offense. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 303–314, 2006.

[114] Michael Walfish, Mythili Vutukuru, Hari Balakrishnan, David Karger, and Scott Shenker. DDoS defense by offense. *ACM Transactions on Computer Systems (TOCS)*, 28(1):3, 2010.

[115] Haining Wang, Cheng Jin, and Kang G Shin. Defense against spoofed ip traffic using hop-count filtering. *IEEE/ACM Transactions on networking*, 15(1):40–53, 2007.

[116] Jiangtao Wang and Geng Yang. An intelligent method for real-time detection of ddos attack based on fuzzy logic. *Journal of Electronics (China)*, 25:511–518, 2008.

[117] XiaoFeng Wang and Michael K Reiter. Defending against denial-of-service attacks with puzzle auctions. In *2003 Symposium on Security and Privacy, 2003.*, pages 78–92. IEEE, 2003.

[118] XiaoFeng Wang and Michael K. Reiter. Defending against denial-of-service attacks with puzzle auctions. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 78–92, 2003.

[119] Brent Waters, Ari Juels, Alex Halderman, and Edward Felten. New client puzzle outsourcing techniques for DoS resistance. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS)*, pages 246–256, 2004.

[120] Wikipedia. Bernstein inequalities (probability theory). `https://en.wikipedia.org/wiki/Bernstein_inequalities_(probability_theory)`, 2024.

[121] Qishi Wu, Sajjan Shiva, Sankardas Roy, Charles Ellis, and Vivek Datla. On modeling and simulation of game theory-based defense mechanisms against dos and ddos attacks. In *Proceedings of the 2010 Spring Simulation Multiconference*, pages 1–8, 2010.

[122] Qiao Yan, Wenyao Huang, Xupeng Luo, Qingxiang Gong, and F Richard Yu. A multi-level ddos mitigation framework for the industrial internet of things. *IEEE Communications Magazine*, 56(2):30–36, 2018.

[123] Andrew Yao. Probabilistic computations: Toward a unified measure of complexity. In *IEEE Symposium on Foundations of Computer Science*, pages 222–227, 1977.

[124] Guang Yao, Jun Bi, and Peiyao Xiao. Source address validation solution with OpenFlow/NOX architecture. In *2011 19Th IEEE international conference on network protocols*, pages 7–12. IEEE, 2011.

[125] Jie Yu, Zhoujun Li, Huowang Chen, and Xiaoming Chen. A detection and offense mechanism to defend against application layer ddos attacks.

In *International Conference on Networking and Services (ICNS'07)*, pages 54–54. IEEE, 2007.

[126] Bin Yuan, Huan Zhao, Chen Lin, Deqing Zou, Laurence Tianruo Yang, Hai Jin, Ligang He, and Shui Yu. Minimizing financial cost of DDoS attack defense in clouds with fine-grained resource management. *IEEE Trans. on Network Science and Eng.*, 7(4):2541–2554, 2020.

[127] Xiaoyong Yuan, Chuanhuang Li, and Xiaolin Li. DeepDefense: Identifying DDoS attack via deep learning. In *Proceedings of the IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–8, 2017.

[128] Chu YuHunag, Tseng MinChi, Chen YaoTing, Chou YuChieh, and Chen YanRen. A novel design for future on-demand service and security. In *2010 IEEE 12th International Conference on Communication Technology*, pages 385–388. IEEE, 2010.

[129] Mahdi Zamani, Jared Saia, and Jedidiah Crandall. TorBricks: Blocking-Resistant Tor Bridge Distribution. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 426–440. Springer, 2017.

[130] S. T. Zargar, J. Joshi, and D. Tipper. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE Communications Surveys Tutorials*, 15(4):2046–2069, 2013.

[131] Saman Taghavi Zargar and James Joshi. Dicodefense: Distributed collaborative defense against ddos flooding attacks. In *Proc. IEEE Symp. Security Privacy*, pages 1–3. Citeseer, 2013.

[132] Menghao Zhang, Guanyu Li, Shicheng Wang, Chang Liu, Ang Chen, Hongxin Hu, Guofei Gu, Qianqian Li, Mingwei Xu, and Jianping Wu. Poseidon: Mitigating volumetric ddos attacks with programmable switches. In *the 27th Network and Distributed System Security Symposium (NDSS 2020)*, 2020.

[133] Shengbao Zheng and Xiaowei Yang. Dynashield: Reducing the cost of DDoS defense using cloud services. In *Proceedings of the 11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, 2019.