Blind Slashing: Moderation of Multi-Party Interaction in Decentralized Systems with Limited Individual Information

Anonymous Author(s)
Submission Id: 142

Abstract

In decentralized cloud computing marketplaces such as the iExec marketplace, ensuring fair and efficient interactions between requesters, asset providers, and computing providers is crucial. Traditional mechanisms often fail to address the complexity of task failures due to limited visibility into individual actor behavior. This paper introduces Blind Slashing, a novel approach designed to address these challenges by implementing a penalty system that broadly affects all involved parties when a task fails, rather than isolating penalties to the computing provider alone. By leveraging a game-theoretic framework, we analyze the strategic behaviors of requesters, asset providers, and computing providers under the current system. Then using ruin theory, we show that Blind Slashing incentivizes improved behavior across all actors by creating a disincentive for faults and misbehavior, leading to a more robust and fair marketplace. This approach not only enhances the reliability of the system but also simplifies the management of task failures, providing a promising solution for decentralized environments where information is inherently limited.

Keywords: decentralized marketplace, smart contracts.

1 Introduction

Key players in today's e-commerce landscape are logically centralized companies (e.g., Amazon, Alibaba, Ebay). This centralization of power renders the marketplace vulnerable to attacks [**?**], failures [**? ?**], and undesirable behaviors such as censorship and bias on goods proposed to consumers [**? ? ?**].

Novel decentralized marketplaces like OpenSea or Rarible aim at solving these problems, notably by using blockchain and decentralized storage (e.g., IPFS [**?**]) as building blocks. In such systems, anyone can offer assets for sale or look up assets to buy. A match-making algorithm matches buy and sell orders emanating from each side of the market.

A key functionality for enabling buyers to find assets they want to acquire is a search mechanism. Unfortunately, current decentralized marketplaces either lack an integrated search mechanism (e.g., OpenBazaar), or use a centralized search mechanism (e.g., OpenSea), thereby reintroducing risks of censorship and bias.

In recent years, solutions were proposed to enable decentralized search mechanisms in decentralized marketplaces [**?**

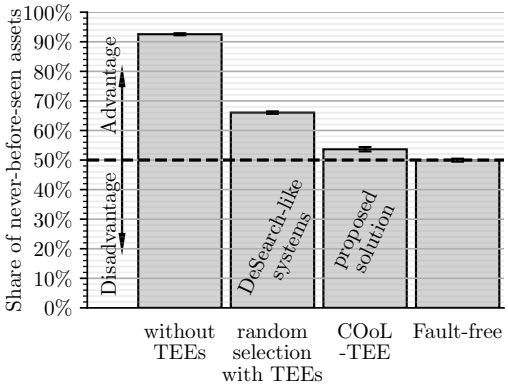


Figure 1. *Information front-running: without TEEs or without latency-aware provider selection, malicious consumers are able to be more often the first to see new assets* | Share of never-before-seen assets discovered first by malicious consumers ($\frac{50}{100}$ of total consumers), under different configurations. $\frac{6}{12}$ search providers are malicious and the total system load is 50%, see results in Section 6 for a more exhaustive analysis.

? ?]. Notably, DeSearch [**?**] proposes a multi-keyword search for decentralized services. It leverages Trusted Execution Environments (Intel SGX [**?**]) to protect buyers from censorship and bias attacks. DeSearch relies on a subset of users contributing their computing assets to run the decentralized search protocol and act as service providers.

In a context where assets are valuable, scarce, and often both, we must ensure that search service providers cannot favor some users and penalize others. We are particularly interested in the case where service providers manipulate access to knowledge about new assets for some users, thereby giving a head-start to others. We call this phenomenon an *information front-running* (IFR) attack. We show in this paper that recent work on decentralized search mechanisms for decentralized marketplaces is vulnerable to IFR attacks: malicious service providers have the power to delay responses to honest consumers, in favor of malicious consumers, which we highlight as a novel attack in the context of TEEs. Malicious consumers can accentuate the providers' attack by loading honest providers, increasing those providers' end-to-end latency, so as to steer honest consumers towards malicious consumers who will then attack them.

We illustrate the impact of IFR in Figure 1: in an open system (1st bar) malicious providers and malicious clients can

gain significant advantage due to their capacity to perform IFR and content attacks. DeSearch (2nd bar) prevents the latter by using TEEs, but is still vulnerable to delay attacks. Blind Slashing (3rd bar) prevents delay attacks and is the closest to the behavior of a fault-free system (4th bar).

We propose Blind Slashing, Client-side Optimization of Latencies coupled with TEEs, a mechanism to protect decentralized search systems from highlighted information front-running attacks. Building upon the DeSearch [**?**] protocol, Blind Slashing adds a *Provider Selection Module* at the client-side, that selects providers based on past experiences with them, with respect to observed request-response round-trip latencies. Consumers are then able to steer away from providers with higher latencies, be it due to network distance or malicious behavior, and send their requests to providers that minimize experienced latencies. The proposed solution is client-side for simplicity and deployability: it does not require server-side modifications, coordination, or consensus.

We evaluate Blind Slashing in both a deployed cluster and a simulated environment (source code available [**?**]), in network setups featuring homogeneous or heterogeneous latencies between consumers and providers. We compare the impact of Blind Slashing and related work in an extensive study, in different system configurations and under different attacker strengths. In particular, we evaluate the competitive advantage (or lack thereof) for a subset of consumers, and overall latency Quality-of-Service. We show that in networks where latency heterogeneity between users is low, with Blind Slashing, malicious users do not get an edge over others in terms of fresher market state information, as long as there is enough honest computing power to serve all requests. We also highlight how high heterogeneity in network latencies between users can give an intrinsic advantage to users who are already close to providers, even without attacks.

The remainder of the paper is structured as follows. We present in Section 2 a model of the ecosystem and actors involved in decentralized marketplace search, upon which we then define our threat model. Sections 3 and 4 provide an overview followed by a detailed description of Blind Slashing. Next, in Sections 5 and 6, we respectively present our experimental protocol and our results. In Section 7, we present related work, before concluding in Section 8.

2 Model & problem statement

We detail the model of decentralized marketplace search that we consider and formalize our system and attacker models.

2.1 Roles & interactions

We assume the system is composed of the following three entities, also illustrated in Figure 2: (i) the *market* (represented in the right part of the figure) is the blockchain- and/or decentralized storage-based system in which assets are registered for future transactions, for example, to be sold or rented.

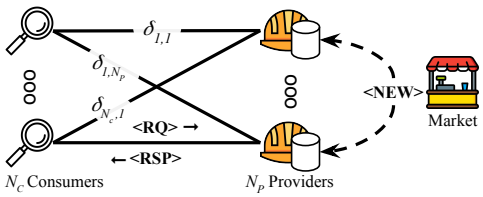


Figure 2. *Marketplace search model* | Consumers send requests to Providers of the search mechanism about the market's state; Providers then respond with a list of assets they previously learned from the Market over time.

contributors of assets create the offer side of the market. In the scope of this paper, contributors and their actions are abstracted as a stream of *market events* towards *providers*; (ii) *consumers* (depicted in the left part of the figure) are users of the search mechanism; and (iii) *providers* (depicted in the middle of the figure) provide their computing power to run the search mechanism: they act as intermediaries between the market and consumers. To do so, providers update a local index of the market state according to the market events they receive, which they use to respond to consumers' search requests.

Some providers may collude with some consumers and offer these better treatments, typically in the exchange of bribes. These actors are considered malicious. The objective of malicious consumers is to maximize the probability that they are the first to see new assets before the rest of the consumer base. We call Never-Before-Seen assets (NBSa) assets that are not yet discovered by any consumer. NBSa become discovered NBSa (dNBSa) by some consumer once a response that contains those NBSa arrives at that consumer. We refer to attacks that impact a consumer's share of dNBSa compared to others' as *information front-running* (IFR) attacks.

2.2 System formalization & constraints

As illustrated by Figure 2, we consider N_C consumers and N_P providers placed on a network topology. The communication delay between a consumer i and a provider j is $\delta_{i,j}$ (also referred to as δ_{net} as a generic notation). Consumers may be co-located with providers, in which case we assume that communication is negligible compared to queuing and computation delays, i.e., $\delta_{i,j} = 0s$.

We assume that two protocols run in parallel: the market and the search mechanism. Messages sent as part of those protocols will be represented in the format < **type** : **data** > (or the short-hand < **type** >). On the market side, we assume that new assets emanate from the market towards providers, at an average rate λ_a. For each new asset, < **NEW** > messages are broadcast to all providers. We assume (for the sake of simplicity) that all providers receive < **NEW** > messages at the same time. The representation of assets in < **NEW** >

messages contains all the metadata necessary for subsequent indexing (e.g., its defining keywords).

Concurrently, as part of the search mechanism, consumers send requests < **RQ** > to providers. Requests contain a list of keywords of interest to the consumer. Each consumer sends requests at a rate λ_r, each request to a single provider. The provider handles those < **RQ** > messages following a First-In-First-Out policy. Specifically, messages wait in the queue for a delay δ_q, depending on the queue size at arrival. When a request < **RQ** > is handled, the provider searches their local index of assets according to the request's keywords. A fixed response time D is required to handle the request. Then, they respond with a message < **RSP** >, containing a list of assets that matched the filters. Note that only assets whose < **NEW** > message arrived before the consumer's request can be present in < **RSP** > messages.

2.3 Building blocks

We now describe the protocols and building blocks forming the foundation of this work.

2.3.1 Trusted Execution Environments. Our work and the DeSearch protocol described below leverage Trusted Execution Environments (TEEs), namely Intel SGX [**?**], to guarantee the integrity and confidentiality of the search protocol's computations and communication.

TEEs are hardware-based security components integrated in CPUs. They enable running code in a protected environment, called an enclave. An enclave is isolated from the rest of the system including the operating system and other applications. This enclave can be remotely attested, e.g., by a consumer, to prove that the code running inside it is the expected one and is indeed running in a genuine TEE environment. Following this attestation, a secure channel can be established between the consumer and the enclave, to ensure confidentiality and integrity of their communications. In our context, this means that the host, which may be malicious and is, therefore, not trusted, is not able to modify the content of the responses sent by the enclave. However, the host is still able to delay the responses, once they are outside the TEE, a property that we exploit in this paper.

2.3.2 DeSearch [?] DeSearch is a decentralized search protocol for decentralized marketplaces, that leverages TEEs to protect consumers from censorship and bias attacks. It is an epoch-based protocol that follows a 3-step pipeline: new assets registered on some verifiable storage, like blockchains or IPFS [**?**], are first crawled; these crawled assets are then indexed in the following epoch; and finally, "Queriers", our providers, serve search requests from clients, our consumers, in the third epoch. Our work focuses on the last step of this pipeline: the index created at the end of DeSearch's second step includes the new assets from the market. This index is then fetched and served by providers. That is, in this paper, we abstract DeSearch's phases upstream of the search mechanism as the market (right side of Figure 2).

DeSearch uses TEEs to guarantee that results are complete and tamper-proof, such that providers cannot censor or bias results. Additionally, it is necessary to hide accesses to the index of assets in memory from the possibly malicious host, as observing memory access patterns may allow it to infer the content of requests and perform targeted censorship. In DeSearch and our work this threat is prevented by the use of Oblivious RAM, specifically Circuit-ORAM [**?**] for accessing the index of assets. Note that while ORAM randomizes memory access patterns, it also prevents the parallelization of serving requests across multiple threads accessing the same shared index [**?**]. As a result, both for DeSearch and Blind Slashing it is assumed that requests are served sequentially.

2.4 Threat model

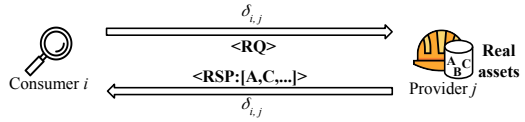
We assume that a fraction c_M of consumers and p_M of providers are malicious in the system. In the following, for the purpose of intuitive exposition, we consider that half the consumer-base is malicious ($c_M = 50\%$) unless otherwise specified. Specific attack behaviors are discussed in later subsections. All other actors are assumed to be honest and they therefore follow the protocol correctly.

We assume that Providers use TEEs to protect the integrity of the search protocol as well as the confidentiality of the request/response messages.

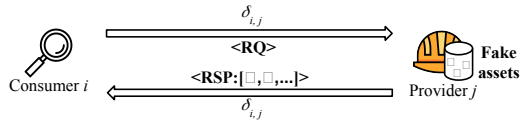
We assume that malicious actors have similar available computing power compared to honest actors with the same role (e.g., between consumers). In total, this computing power is proportional to c_M and p_M. We assume that honest and malicious consumers have a respective request budget proportional to their fraction of the total population, e.g., proportional to c_M for malicious consumers.

2.4.1 Adversarial objective. We assume that malicious providers gain more when malicious consumers can acquire assets instead of honest ones (e.g., through bribes). The common objective for malicious actors is to maximize their share of discovered Never-Before-Seen assets (NBS-assets) compared to honest consumers. We define that an asset A was discovered first by a given consumer if a response < **RSP** > containing that asset was received by that consumer before any other consumer. The strength of the *information front-running* experienced by the malicious faction in a given time frame is measured by the fraction of NBS-assets discovered first by malicious consumers, compared to the total number of new assets. The higher above c_M this fraction is, the bigger the advantage malicious customers have over honest ones in future market transactions.

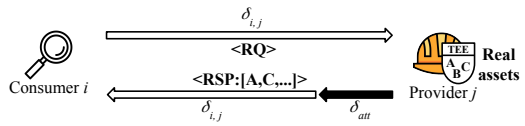
Given that both honest and malicious consumers compete over the same set of assets, preventing honest consumers from discovering NBS-assets first is a viable strategy for the



(a) Fault-free | Consumer and Provider exchange a request for a response containing a list of assets.



(b) Content attack | Without TEEs, a malicious provider can send back fake or old assets to hinder asset discovery by honest consumers.



(c) Timing attack | Content attacks (Figure 3b) are not feasible with TEEs, but providers can still delay responses so honest consumers receive staler market information than malicious ones.

Figure 3. Provider-side attack models on decentralized marketplace search

malicious faction. The means available to the malicious faction are discussed in the following subsections and illustrated by Figure 3.

2.5 Attacks formalization

Based on the system constraints and the malicious faction's objective, we now define attacks led by malicious actors.

2.5.1 Content attacks. To compare the IFR attack impact in systems with and without TEEs, a strawman approach without TEEs is considered where malicious actors can have the most impact on the system. Indeed, without TEEs, malicious providers are able to read requests plainly and to modify (e.g., to fake) the content of the corresponding responses (Figure 3b), instead of sending only real assets as in the fault-free case (Figure 3a). Consequently, in the worst case, if an honest request is served by a malicious provider, it cannot be used to discover new assets. As a result, for the same number of sent requests, malicious consumers have more opportunities to discover new assets than honest consumers. We refer to this worst-case as *content attacks*.

2.5.2 Timing attacks. While TEEs uphold the integrity and confidentiality of the search protocol, TEEs do not prevent malicious providers from delaying responses to honest consumers. This is a novel attack that TEEs cannot solve by themselves, where the surrounding host may arbitrarily delay messages coming in or out of the TEE, based on the message sender or recipient. Indeed, malicious providers can delay sending responses back to honest consumers by some delay δ_{att} (Figure 3c), while they respond to malicious consumers immediately after computation, like the fault-free behavior (Figure 3a). Due to the TEE's computation integrity and its secure channel with the consumer, the response is correct and tamper-proof. Therefore, δ_{att} should be applied after the response is computed to lower its freshness. We refer to this behavior as a *timing attack*. We consider that malicious providers do not drop < **NEW** > messages. With DeSearch [**?**], search results are verifiably complete each epoch.

2.5.3 Cuckoo attacks. In this attack, malicious consumers target honest providers by overwhelming them with requests in order to slow them down. Consequently, honest consumers who choose providers with lower latencies will leave honest providers in favor of malicious providers, who give them staler information through timing attacks. We call this consumer-side attack a *cuckoo-timing* attack. We also evaluate cuckoo attacks on top on content attacks, i.e., *cuckoo-content attacks*, for comparison purposes.

3 Blind Slashing | Overview

In Section 3.1, we focus on the timing aspects we consider for the two protocols that operate in parallel in the considered decentralized marketplace search mechanism, as illustrated in Figure 4: the market indexing mechanism (right), as well as our proposed provider-selection and search mechanisms (left). We then describe the key principles of the main module involved in Blind Slashing's search mechanism in Section 3.2: the *Provider Selection Module* (PSM).

3.1 Protocol delays and timing

We focus here on the timing of our protocol steps. Figure 4 shows the behaviour of a consumer i that uses the search mechanism and providers 1 to N_P who provide this service: (step 1) at t_{gen}, the consumer i, on the left of Figure 4, first generates a request message < **RQ** > . (2) The request is directed by the PSM to some provider j at t_{send}, after a computation delay δ_{PSM}. (3) The request is sent through the network to the chosen provider j, with a delay $\delta_{i,j}$. The untrusted host then receives the request message and forwards it to the TEE, who enqueues it. (4) The request waits for δ_{queue} in the TEE's queue until service of this request begins at t_{serv}. The time to serve the request after t_{serv} is $\delta_{serv} = D$. (5) After service completion, the TEE sends the response message < **RSP** > out to the untrusted host, who, if it is malicious, may delay the response by δ_{att}. The response is then sent through the network to the consumer i, with a return-trip delay $\delta_{i,j}$. (6) At t_{recv}, the consumer i finally receives the response message < **RSP** > and the PSM logs the experienced round-trip latency, for future interactions. Let k be the number of request-response interactions between consumer i and provider j. Consumer i logs $\Delta^k_{i,j} = t_{recv} - t_{send} = \delta_{queue} + D + 2\delta_{i,j} + \delta_{att}$.

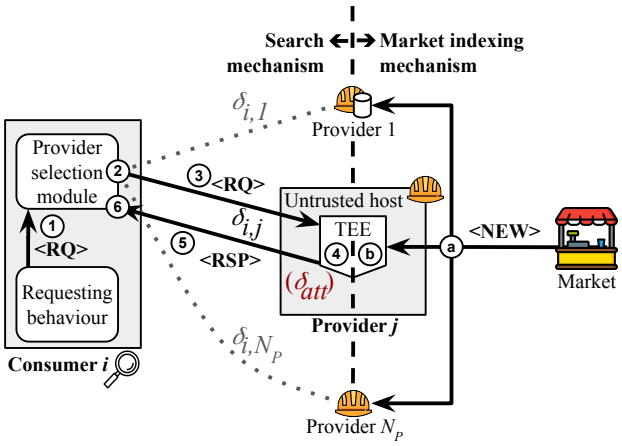


Figure 4. High-level system architecture

Concurrently, on the right of Figure 4, as a new asset A appears on the market, < **NEW** > messages are broadcast to all providers (step a). Then, in step b, we consider that A is added to the index at the same t_{index} for all providers. For A to appear in a given < **RSP** > message, whose request would match A, A must be indexed before serving the request: we require $t_{index} < t_{serv}$. Furthermore, due to the TEE's computation integrity, an indexed asset A will appear in responses if it matches the request. Therefore, from a consumer's point of view, a response's freshness with respect to the market's state is the latency once service starts: $\delta_{fresh} = D + \delta_{i,j} + \delta_{att}$. Honest consumers can maximize the freshness (i.e., minimize δ_{fresh}) by choosing close providers in the network (low $\delta_{i,j}$), with high computation power (low D), and who are honest ($\delta_{att} = 0$).

3.2 COoL | Client-side Optimization of Latencies

The key mechanism in Blind Slashing is the Provider Selection Module (PSM). Before providing a detailed description, we first introduce the PSM's key principles. A consumer wants the freshest information about the market, that is, responses that contain the newest relevant assets. As a secondary goal, consumers also want to minimize their request-response latencies, for user experience purposes.

As a consequence, the goal of a consumer's PSM is to prefer sending requests to providers with whom the consumer experiences low round-trip latencies, and in particular to avoid malicious providers leading timing attacks ($\delta_{att} > 0$). Each consumer i's PSM does so individually, by performing a provider selection based on *Client-side Optimization of Latencies* (COoL): it aggregates each past request k's round-trip latency $\Delta^k_{i,j}$ with each provider j, then tunes the probability to select each provider for future requests, so as to decrease the average round-trip request latency across all providers over time. Considering a target throughput λ_r of requests to send, consumer i divides this throughput such that each provider j receives a share $\lambda_{i,j} = r_{i,j}\lambda_r$ of the requests (with $\sum_{j\in[\![1,N_P]\!]} r_{i,j} = 1$). At step 2 of the search mechanism (see Figure 4), the goal of the *Provider Selection Module* is to tune all $r_{i,j}$ such that the average round-trip latency is minimized. Doing so, Blind Slashing's PSM indeed chooses providers with low round-trip latencies. Moreover, if an honest and a malicious provider j and j' have the same service time D and network latencies $\delta_{i,j} = \delta_{i,j'}$ to consumer i, then consumer i will preferentially choose the honest provider, whose $\delta_{att} = 0$.

4 Blind Slashing | Detailed description

The key to selecting providers such that average round-trip latencies are minimized lies in the design of the *Provider Selection Module*, which is detailed in this section.

4.1 Available information

The *Provider Selection Module* has access to the sequences of observed round-trip latencies for each provider j, denoted $\{\Delta_j = [\Delta^k_j] | j \in [\![1, N_P]\!]\}$, where Δ^k_j is the round-trip latency of the k-th request-response exchange between the *Consumer* and *Provider* j. It also maintains the set of selection ratios, i.e, the probabilities to select each provider j, denoted $\{r_j | j \in [\![1, N_P]\!]\}$, such that $\sum_{j\in[\![1,N_P]\!]} r_j = 1$.

4.2 Module design rationales

As the incoming request load on a given provider increases, the queuing delay at its TEE increases, which in turn increases the round-trip latency for the requests it serves. This means that a consumer may need to divide its target request throughput among several providers to minimize the average round-trip latency it experiences. With this objective in mind, and based on currently available information, the *Provider Selection Module* shall tune the selection ratios $\{r_j | j \in [\![1, N_P]\!]\}$. In other words, given a provider j and observed round-trip latencies Δ_j for this provider, as well as some scoring function f, if $f(\Delta_j) < \text{avg}_{k\in[\![1,N_P]\!]}(f(\Delta_k))$, i.e., if provider j under-performs compared to the average of all providers, then r_j should be decreased. Conversely, a higher-than-average score should increase the selection ratio r_j. In the following, we will use a sliding-window average of size s as the scoring function f, that is, we average the s latest observed latencies.

4.3 Submodules description

The provider selection operates in two distinct operations, the selection itself each time a request should be sent, at step 2 in Figure 4, and the update of the selection ratios, which may happen asynchronously to the search mechanism. In the implemented solution, the update of the selection ratios happens each time s responses are received, with s the sliding window size used in the ratio update algorithm.

4.3.1 Provider selection. At step 2 of the search mechanism (see Section 3 and Figure 4), the *Provider Selection*

Module selects a provider j with a probability r_j, where the request will then be sent in step 3.

4.3.2 Client-side Optimization of Latencies.

In a recurring fashion, e.g., each time s responses are received, the *Provider Selection Module* updates the selection ratios $\{r_j\}$, based on the observed round-trip latencies $\{\Delta_j\}$, using Algorithm 1.

Algorithm 1: UpdateSelectionRatios

Input: $\{r_j^{in} | j \in [\![1, N_P]\!]\}$ the initial set of selection ratios for each provider j, such that $\sum_{j \in [\![1,N_P]\!]} r_j = 1$; $\{\Delta_j = [\Delta_j^k] | j \in [\![1, N_P]\!]\}$ the set of sequences of observed round-trip latencies (ordered from newest to oldest latencies) for each provider j;

Data: x the exploration coefficient; K_p the PD-controller error-proportional coefficient; K_d the PD-controller error-derivative coefficient; s the sliding window size

Output: $\{r_j^{out} | j \in [\![1, N_P]\!]\}$ the new set of selection ratios for each provider j, such that $\sum_{j \in [\![1,N_P]\!]} r_j = 1$;

1 **begin**
2 $\quad \Delta_j^{avg} \leftarrow \frac{1}{s} \sum_{k=1}^{s} \Delta_j^k \quad \forall j \in [\![1, N_P]\!]$;
3 $\quad \Delta_j^{prev,avg} \leftarrow \frac{1}{s} \sum_{k=s+1}^{2s} \Delta_j^k \quad \forall j \in [\![1, N_P]\!]$;
4 $\quad \Delta^{avg} \leftarrow \frac{1}{N_P} \sum_{j=1}^{N_P} \Delta_j^{avg}$;
5 $\quad$ **for** $j \in [\![1, N_P]\!]$ **do**
6 $\quad\quad e_j \leftarrow \frac{\Delta^{avg} - \Delta_j^{avg}}{\Delta^{avg}}$;
7 $\quad\quad d_j \leftarrow \Delta_j^{avg} - \Delta_j^{prev,avg}$;
8 $\quad\quad r_j^{tmp} \leftarrow clamp(r_j^{in} + K_p e_j + K_d d_j, 0, 1)$;
9 $\quad$ **for** $j \in [\![1, N_P]\!]$ **do**
10 $\quad\quad r_j^{norm} \leftarrow \frac{r_j^{tmp}}{\sum_{j \in [\![1,N_P]\!]} r_j^{tmp}}$;
11 $\quad\quad r_j^{out} \leftarrow (1 - x) r_j^{norm} + x \frac{1}{N_P}$;
12 $\quad$ **return** $\{r_j^{out} | j \in [\![1, N_P]\!]\}$

In Algorithm 1, we use the logic of a PD-controller (Proportional Derivative) to update the selection ratios, with the error-proportional coefficient K_p and the error-derivative coefficient K_d. The setpoint, i.e., the target value, for this controller is the average score among all providers, i.e., the average round-trip latency among all providers, Δ^{avg}, in the last s request-response interactions with each provider. The objective of the controller is to incrementally update selection ratios r_j to bring provider-specific average latencies Δ_j^{avg} closer to the global average latency of all providers Δ^{avg}.

In more detail, the algorithm proceeds as follows, and as illustrated by Algorithm 1. First, average round-trip times for the current and previous windows are computed for each provider j (lines 2 and 3), as well as the global average round-trip time Δ^{avg} (line 4), the setpoint of the PD-controller. Then, for each provider j, the error e_j with respect to the setpoint and the derivative of the error d_j are computed (lines 6 and 7). e_j is the relative difference between the provider j's average latency and the global average latency. d_j is the difference between the provider j's average latency in the current window and the previous window. Then, the selection ratio r_j is updated using the PD-controller formula, and clamped so the resulting value is the nearest available in the interval $[0, 1]$ (line 8). The selection ratios are normalized (lines 10), i.e., $\sum r_j = 1$. Finally, we remark that this client-side provider selection can be described as a Multi-Armed Bandit problem [**?**]: a consumer sends a request to a chosen provider j (i.e., pulls lever j), and experiences a round-trip latency (i.e., the reward). The consumer's objective is to maximize its reward: select providers such that it minimizes latencies. As is common in Multi-armed bandit settings [**?**], the selection policy should allocate a small fraction of requests to explore the latency-performance of each provider, to avoid exploiting sub-optimal providers asymptotically. This is achieved by mixing a uniform probability of selecting each provider $\frac{1}{N_P}$ with the current values of r_j, weighted by x and $(1 - x)$ respectively (line 11).

We draw attention to the fact that the presented algorithm requires calibration of the coefficients K_p and K_d, as well as of the exploration coefficient x, so as to offer theoretical guarantees. This can be online, during the system's operation, with parameter-tuning algorithms [**? ?**]. Algorithms from the Multi-Armed Bandit literature that tolerate evolving reward distributions, like Exp3 [**?**], may also be used.

As our focus was not on convergence, but on the asymptotic behavior of the system where selection ratios have converged, parameters were tuned using the Ziegler–Nichols method [**?**]. We tune the system such that its convergence was near-optimal in micro-benchmark scenarios, i.e., in a set of stereotypical network scenarios (e.g., high- versus low-latency providers; high- versus low-throughput providers), and converged with random topologies drawn from the dataset which will be presented in the next section.

5 Experimental protocol

We present in this section the performance evaluation of Blind Slashing, performed in terms of IFR advantages, as well as overall Quality-of-Service regarding roundtrip search latencies. In particular, this section allows answering the following questions:

RQ1: What is the performance of Blind Slashing in the fault free case?
RQ2: What is the performance of Blind Slashing in an adversarial setting where providers are equally distant

in terms of network latency from consumers (e.g., all providers are deployed in the same datacenter)?

RQ3: What is the performance of Blind Slashing in an adversarial setting where providers are deployed over the internet, i.e., with heterogeneous network distance from consumers?

We answer these questions in Sections 6.1 to 6.3, respectively.

5.1 Metrics

Two types of metrics are used to assess the performance of Blind Slashing: discovered Never-Before-Seen assets (dNBSa) and latencies. The main metric is the number of discovered Never-Before-Seen assets (dNBS-assets) by malicious consumers, as defined in Section 2.

If the proportion of dNBS-assets discovered by any subpopulation of consumers is similar to the proportion of requests sent by that subpopulation, then the subpopulation has no IFR advantage over other subpopulations. If that proportion is higher (resp. lower), then the subpopulation has a higher (resp. lower) IFR advantage (resp. disadvantage) over other subpopulations.

The secondary metric is the latencies experienced during the lifetime of consumer requests and their responses from providers. Indeed, latencies at specific steps of the protocol have an impact on the first metric. Notably, the freshness of a response, i.e., the time elapsed between the moment the service of a request starts and the moment the response is received by the enquiring consumer, is the target of *timing attacks*. Therefore, we also observe the latencies of search protocol phases and those introduced by attacks, to understand their impact on NBS-asset discovery.

Roundtrip latencies will also be considered from the point-of-view of the Quality-of-Service (QoS) experienced by consumers, with lower latencies using a given mechanism indicating better latency-QoS.

5.2 Deployment and simulation setups

We implement the Blind Slashing protocol in C++, using DeSearch [**?**] as the back-end search mechanism. Search providers run on an SGX-enabled 4-core Azure *DC4s_v2* VM, while consumers run on a non-SGX 16-core Azure *B16als_v2* VM. Both machines are in the same datacenter, with a 1.1ms (0.5ms of standard deviation in 100 pings) round-trip latency between them. We use this deployment to measure the overheads incurred by our protocol in a distributed setting, and to calibrate parameters of a simulated version of the system.

Indeed, due to the high number of configurations to evaluate, we also implement a simulated version of Blind Slashing and of baselines: all experiment design points presented in this paper represent around 40k protocol runs and 1TB of requests' lifecycle traces.

Protocols are simulated using Omnet++ [**?**], a discrete event simulator. Only the protocols involved during requests' lifecycle are simulated: in practice, TEE remote attestation can be performed as an independent first phase between each consumer-provider pair. The search phase, which consists of many request-responses interactions, may start after this attestation. Therefore, remote attestation is left out of simulations. During the search phase, as in our assumptions in Section 2, providers receive the same **<NEW>** messages at the same time. Additionally, we do not consider the delay introduced by handling these messages, i.e., they are available instantly when they arrive at providers, to be used in subsequent requests.

The code for the distributed and simulated implementations, as well as the analysis scripts, are available publicly [**?**].

5.3 Experiment configurations and baselines

Based on the implementation and networking dataset, we define experiment configurations that differ across the following dimensions.

5.3.1 Simulation parameters. Simulation experiments are run with a total of $N_C = 100$ consumers and $N_P = 8$ providers. Providers each serve requests at a rate of 160 requests per second, i.e., they take $D = 6.25ms$ to handle a request. This maximum throughput value was determined based on performance experiments of DeSearch [**?**] in our deployment environment.

Consumers send requests at equal rates λ_r, such that the total request rate is $\lambda_T = N_C\lambda_r = \frac{\rho}{D}N_P$, with ρ the target overall system load. Inter-request emission delays are exponentially distributed with parameter λ_r.

Assets arrive on the market at an average rate of $\lambda_a = 100$ assets per second, according to an exponential distribution $Exp(\lambda_a)$.

We also discuss results of experiments run either with periodic asset arrivals, or periodic inter-request emission delays, but they will not be illustrated in the paper.

5.3.2 Network topologies. We consider two network topology types: (1)"same datacenter" topology, where latencies are considered equal between all actors and (2) "PlanetLab" topology, where latencies are heterogeneous and assigned using the PlanetLab dataset [**?**]. This dataset represents the average roundtrip latencies between 490 nodes of the PlanetLab network, positioned across the globe.

5.3.3 Attack scenarios. In the following experiments, malicious providers' proportion p_M is in $[0, 1]$, while malicious consumer's c_M is set to 50%, for more intuitive result analysis: in the fault-free case without malicious providers, neither the malicious or the honest consumers should have an advantage over the other if they are present with equal proportions.

The protocol can be attacked according to the strategies defined in Section 2. Experiments configurations are labelled

with the name of the attack strategy, or "fault-free" if no attack is performed.

Timing attacks require a malicious artificial delay parameter δ_{att}, which we set to $\delta_{att} = 50ms$ in the experiments with *timing attacks*. This choice was made based on calibration experiments and will be discussed in Section 6.3.

5.3.4 Provider selection policies.

Consumers select providers to handle an individual request using one of the following policies: a uniformly random selection of a provider among the N_P providers, referred to as "DeSearch-like" selection, or a selection based on Blind Slashing's policy, locally optimizing the selection of providers to minimize end-to-end latencies, referred to as "COoL" selection.

At the provider-side, in the same-datacenter setup, we also evaluate the impact of Blind Slashing with the state-of-the-art Power-of-Two (PoT) load balancing protocol [**?**]. In particular, we evaluate spatial Power-of-Two [**?**] (sPoT), which selects the least-loaded of a consumer's two closest providers. We also consider a variant of Blind Slashing augmented with PoT, which we call, which we call "COoL-PoT". In COoL-PoT, a consumer's *Provider Selection Module* randomly selects $k = 2$ distinct providers (proportionally to their selection ratios r_j). Both providers will exchange with each other their respective queue lengths at the request's arrival: the provider with the shortest queue handles it, while the other drops the request. Ties are broken deterministically, e.g., based on the lowest-valued hash of the provider's ID and the request's nonce (similarly to rendezvous hashing [**?**]).

Note that this new mechanism introduces a PoT specific attack, which we refer to as a "Queue attack" (or "Queue-T attack" when combined with timing attacks). In a queue attack, malicious providers keep a secondary queue outside the TEE, so that the TEE's queue always remains at a length $L \leq 1$ (considering a request being handled remains at the front of the queue until served). This means that from the point of view of honest providers, malicious providers more often have shorter queues than them and consequently, will yield service of a request to them more often.

We evaluate the state-of-the-art solutions that rely on sending the same request to multiple providers as a class of "multiprovider" selection policies, with k the number of providers selected for the same request content. Only honest consumers perform this multiple provider selection, aiming to reduce IFR disadvantages. Malicious consumers only send requests to one destination (i.e., $k = 1$ in all cases), because they are treated the same way by all providers.

6 Results

We now present experimental results, aiming to answer our research questions formulated in the previous Section 5.

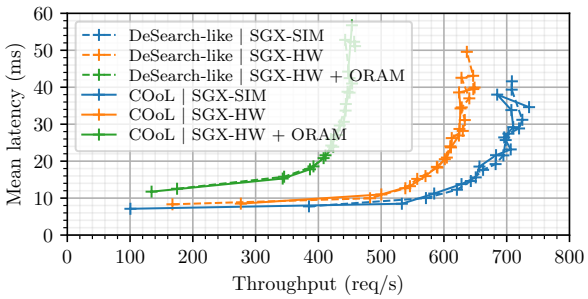


Figure 5. Latency vs. throughput for DeSearch-like and Blind Slashing provider selection (among $N_P = 4$ providers), based on enabled security features (SGX simulation/hardware mode and Circuit-ORAM [**?**])

6.1 Fault-free setup

To answer *RQ1*, we first evaluate the impact of the provider selection policy in a fault-free setting. We compare building block overheads on the same-datacenter topology, and the latency Quality-of-Service of selectio policies in the PlanetLab topology, with heteregeneous network latencies between consumers and providers.

6.1.1 Building block overheads.

Figure 5 illustrates overheads incurred wrt. the selection policy and enable security features in a fault-free same-datacenter environment. We measure latency from the time when the request is generated to when the response arrives back at the consumer, i.e., we include the latency due to provider selection algorithm. Following the notations in Section 3.1, we therefore measure: $t_{recv} - t_{gen} = \delta_{PSM} + \Delta^k_{i,j}$. The provider selection policy has a negligible impact on the system's maximum throughput (less than 2% in each SGX security setup), as well as on the average latency for a given throughput. This can be explained by the fact that the selection policy is performed at the consumer-side, and so does not impact the provider-side processing throughput. Moreover, selection ratio updates are performed by Blind Slashing outside of a request's lifecycle (e.g., after reception of 10 responses), and therefore does not impact the request-response latency. For both selection policies, the actual selection amounts to a weighted random selection of providers, with the weights being the selection ratios (weights are equal for DeSearch-like systems).

Regarding SGX and ORAM security features, compared to "Simulation" mode without ORAM, using SGX "Hardware" mode causes a 10% throughput reduction and adds a 1ms latency overhead, while also activating ORAM reduces the maximum throughput by 37.5% and adds 5ms of latency.

6.1.2 Latency Quality-of-Service.

We now measure latencies as $t_{recv} - t_{send}$. In terms latency-QoS, Blind Slashing with COoL selection outperforms DeSearch-like selection, as illustrated by Figure 6: COoL selection's median

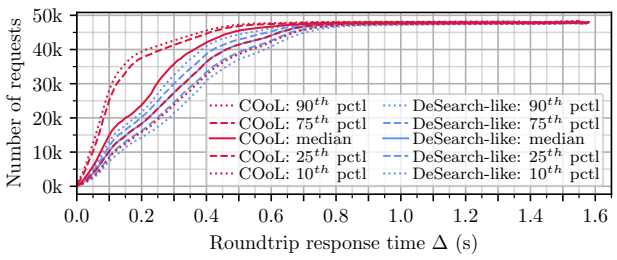


Figure 6. *COoL-TEE's latency-aware provider selection enables lower end-to-end response times than DeSearch-like provider selection* | Cumulative response time quantile distributions, aggregated from 500 runs, in a fault-free environment on the PlanetLab network topology, without TEEs, and a total system load $\rho = 25\%$, for different selection policies.

roundtrip response time is better than the 90% slowest response times with DeSearch-like selection. Meanwhile, DeSearch-like selection's median response time performs similarly to COoL selection's 25% slowest response times. COoL selection policy therefore enables a better latency-QoS than DeSearch-like selection, in a fault-free environment.

6.2 Adversarial same-datacenter setup

We now run simulations in the same-datacenter setup, for the considered attacks scenarios and provider selection policies, with a varying proportion of malicious providers p_M. We also evaluate in this section the impact of scaling out the number of honest providers, and the impact of the Power-of-two load balancing policy [**?**] at the provider-side.

We evaluate information front-running impacts for different configurations, and divide our analysis in three case studies: (1) between single- and multiprovider selection policies; (2) between COoL- and spatial Power-of-two [**?**]-based selection policies; (3) a focus on COoL provider selection subject to the different attacks defined in Section 2.

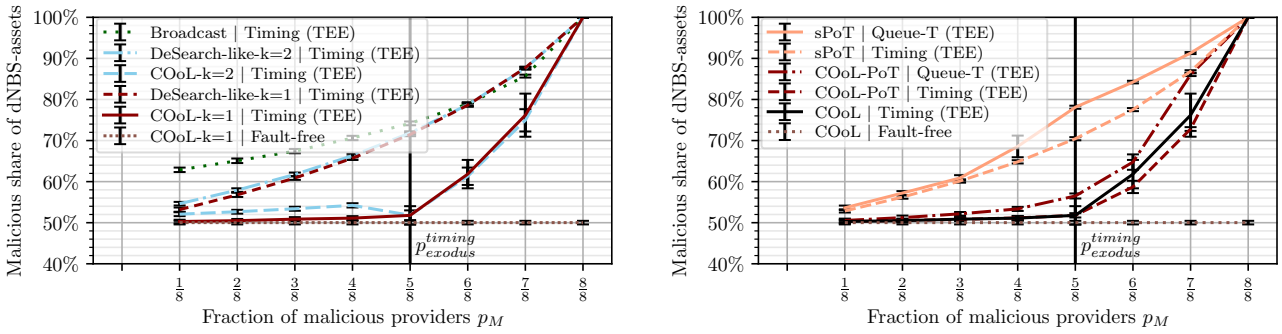
6.2.1 Multiprovider selection. We compare multiprovider selection with either DeSearch-like or COoL policies, with varying $k \in \{1, 2, 8\}$, the number of selection providers per request. In this case-study's 8-provider setup, $k = 8$ corresponds to a broadcast policy used by a subset of related work protocols [**? ?**], and is labeled as such in subsequent figures. Figure 7a illustrates the information front-running advantage experienced by malicious consumers, that is, their share of dNBSa, for *timing attacks* and in the *fault-free* case. For all policies, the higher the number k of honest request replicas, the higher the malicious advantage. However, single-provider COoL selection ($k = 1$) outperforms others: as long as $pM \leq p^{timing}_{exodux}$, information front-running (IFR) advantage remains within 2% of the fault-free case at 50%. p^{timing}_{exodux} is the proportion of malicious providers after which there is not enough honest provider throughput to serve all honest requests, in a context with only *provider-side timing attacks*. Indeed, without cuckoo attacks by consumers, consumers tend to select providers of their faction (either honest or malicious). When $pM > p^{timing}_{exodus}$, this homogeneity is not possible anymore: more and more honest requests are served by malicious providers. In contrast to single-provider COoL selection, DeSearch-like provider selection with any k, as well as COoL multiprovider selection with $k > 2$, show a steady information front-running advantage increase wrt. p_M.

6.2.2 Provider-side load balancing. Now, in Figure 7b, we compare the state-of-the-art spatial Power-of-two [**?**] (sPoT) to our proposed solution, first in a *timing-attacked* TEE environment. We then also evaluate the impact of the new attack introduced by provider-side load balancing: the *queue attack* coupled with *timing attacks*.

First, with *timing attacks* only and with standalone sPoT [**?**], the share of dNBS-assets by malicious consumers increases steadily with p_M. In sPoT, consumers rely on a network distance measurement (e.g., a ping) to determine their two closest providers. In an adversarial same-datacenter setup, where all providers are roughly equidistant to a given consumer, and can lie about one-shot latency measurements, sPoT amounts to single-provider random selection: "DeSearch-like-k=1" in Figure 7a and "sPoT" in Figure 7b have similar shapes. Meanwhile, with consumer-side COoL selection, as well as with its $k = 2$ variant augmented with sPoT at the provider-side, the malicious dNBS-asset share remains close as long as $p_M \leq p^{timing}_{exodus}$.

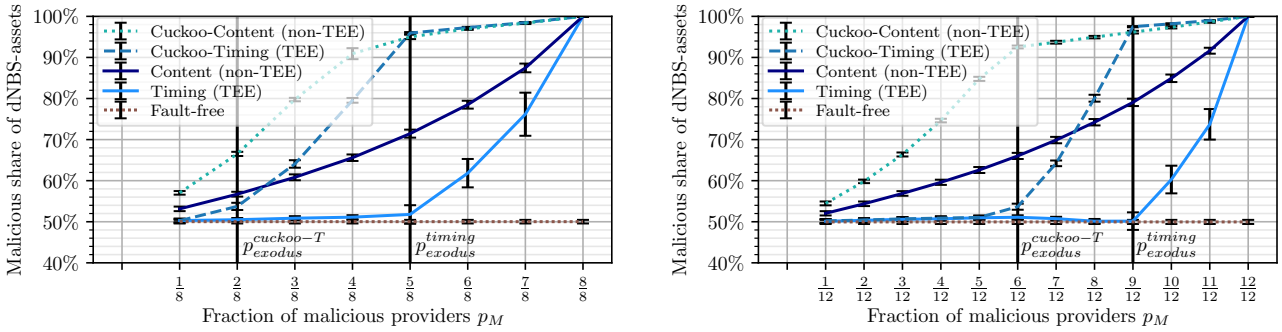
However, when we consider the increased attack surface introduced with provider-side load balancing, i.e., when malicious providers leverage *queue attacks* on top of *timing* attacks, solutions based on this provider-side balancing suffer more from IFR. Standalone "COoL" selection is client-side only, so this attack vector does not exist.

6.2.3 COoL selection under fire. Finally, Figure 8a focuses on single-provider COoL selection Information Front-running (IFR) advantage, when subject to different provider- and consumer-side attacks, still from the point-of-view of dNBSa by malicious consumers. *Content-* and *timing-based attacks* show different behaviors: while (cuckoo-)content attacks have the share of dNBSa grow steadily with p_M, *(cuckoo-)timing attacks* have this share close to the fault-free case, so long as p_M is less than the attack-specific threshold p_{exodus}. After their p_{exodus}, the IFR advantage rises sharply for each *timing-based attack*. Note how $p^{cuckoo-T}_{exodus} < p^{timing}_{exodus}$. Without cuckoo attacks, malicious consumers try to satisfy their latency-minization objective, which drives them towards malicious providers (whom honest consumers avoid). However, with *cuckoo-timing attacks*, malicious consumers give up this objective in favor of actively loading honest providers. This why the honest consumers' shift towards malicious providers happens with a lower p_M, i.e., $p^{cuckoo-T}_{exodus} <$



(a) Impact of TEE *timing attacks* on single-/multi-provider selection **(b)** Impact of TEE-based attacks on sPoT [?] and COoL selections

Figure 7. Information front-running case-study wrt. competitors in a *timing-attacked* TEE environment | Shares of discovered NBS-assets (averages with standard deviation), aggregated from 100 runs, in a malicious environment in the same datacenter setup with 8 providers, for different system configurations. *The closer the values are to 50%, the better, as it indicates lower malicious information front-running advantage.*



(a) Impact of (non-)TEE-based attacks on COoL provider selection with 8 total providers **(b)** Impact of (non-)TEE-based attacks on COoL provider selection scaled out with 4 additional honest providers

Figure 8. Information front-running case-study wrt. attack scenarios in (non-)TEE environments | Shares of discovered NBS-assets (averages with standard deviation), aggregated from 100 runs, in a malicious environment in the same datacenter setup, with either 8 or 12 providers (resp. Figures 8a and 8b) under different attack scenarios. *The closer the values are to 50%, the better, as it indicates lower malicious information front-running advantage.*

p^{timing}_{exodus}. Here, in Figure 8a, the impact of *cuckoo-timing attacks* for a given p_M is similar to a *provider-side-only timing attack* with 3 more malicious providers. In other words, a lower p_M is required for the same IFR advantage, if malicious consumers are willing to trade their latency-minimization objective for an active participation in attacks.

Timing attacks require heavily loading honest providers: we evaluate in Figure 8b the impact of a scale-out behavior by honest providers in response to high loads: 4 additional honest providers are launched compared to Figure 8a. The incoming consumer load remains the same, such that the total system load ρ goes down from 75% (with 8 providers in Figure 8a) to 50% (12 providers in Figure 8b). Lessening the total system load ρ increased the attack-thresholds $p^{cuckoo-T}_{exodus}$ and p^{timing}_{exodus}: while impactful attacks were possible for $p_M > 25\%$ in the 8-provider setup loaded at $\rho = 75\%$, $p_M > 50\%$ is now necessary in the 12-provider setup loaded at $\rho = 50\%$. Indeed, $p^{cuckoo-T}_{exodus}$ is tied to ρ, in that it marks when honest consumers cannot shoulder the total request load anymore, which gives: $p^{cuckoo-T}_{exodus} = 1 - \rho$. p^{timing}_{exodus} depends on ρ, but also on c_M, because only honest consumers overload honest providers in provider-side-only timing attacks, such that: $p^{timing}_{exodus} = 1 - (1 - c_M)\rho$. Note how without TEEs, i.e., under (cuckoo-)content attacks, malicious dNBSa shares are not sensitive to the system load ρ: malicious IFR advantage is a function of p_M for provider-side content attacks, and of p_M and c_M with cuckoo-content attacks. With cuckoo-content attacks, the slope change in Figures 8a and 8b happens when the proportion of malicious providers is equal to the proportion of honest consumers, that is: $p^{cuckoo-C-satur}_{exodus} = 1 - c_M$.

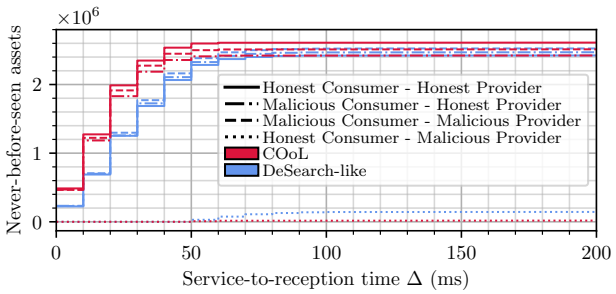


Figure 9. *The staler the response, the less likely NBS-assets will be discovered from it* | Number of NBS-assets discovered with a latency from service start to response reception of Δ milliseconds, depending on the behavior (honest or malicious) of the sending consumer and handling provider, in a TEE-protected environment, with a total system load $\rho = 25\%$, subject to *timing attacks* ($p_M = \frac{4}{8}$ providers are malicious), on the PlanetLab network topology.

6.2.4 Case-studies summary. To summarize and answer *RQ2*, in the same-datacenter setup, with COoL provider selection, as long as the honest providers' maximum throughput is sufficient to serve all requests (or all honest requests under *provider-side-only attacks*), *timing-attacked* honest consumers discover NBS-assets at similar rates as malicious consumers. In practice, this can be ensured using load-aware elasticity mechanisms [**?**]: the third case-study in Section 6.2.3 and Figure 8 show the impact of tuning the system load ρ on the necessary malicious provider proportion p_M for impactful attacks. Meanwhile, without TEEs, or without latency-aware provider selection, dNBSa by malicious consumers increase nearly linearly wrt. the proportion of malicious providers p_M.

Experiments were also run either with periodic asset arrivals, or periodic inter-request emission delays, but do not illustrate due to lack of space: conclusions drawn above still hold in those cases, albeit with higher standard deviations.

6.3 Adversarial PlanetLab setup

The choice of δ_{att}, which will in the end impact the probability that honest consumers discover never-before-seen (NBS) assets thanks to malicious providers, depends on the network topology. Indeed, the lower the response time to a request, from the moment the response is computed to its reception by the consumer, the higher the expectancy on the number of discovered assets before other consumers thanks to that request. Conversely, a high response latency is correlated with rarer NBS-asset discoveries. This response time is affected by the actual network latency δ_{net}, but also by the artificial latency δ_{att}, when malicious providers attack honest consumers. Observing the number of NBS-assets discovered by requests depending on the response time, as illustrated by Figure 9, in a *timing-attacked* system, shows that an overwhelming majority of NBS-assets are discovered within a 50ms response time, and nearly all within 100ms. The dotted lines, dNBSa by honest consumers through malicious providers, are flat at 0 until 50ms, because $\delta_{att} = 50ms$: no response can arrive before that. A very slight increase exists after 50ms, due to honest consumers that were collocated or close to malicious providers in some runs, but it remains negligible compared to what other combinations of consumer and provider behaviors were able to discover. The value of δ_{att} in experiments was chosen based on this observation.

Regarding dNBS-assets by honest and malicious consumers, due to lack of space, we illustrate the results in heterogeneous-latency networks by comparison with Figures 7a and 8a's same-datacenter setup results. Compared to the same-datacenter setup, the observed shares of dNBS-assets discovered by a given consumer type (honest or malicious) are similar for any provider selection under *content attacks*, "DeSearch-like-k=1" selection under *timing attacks*, and in the fault-free case, but with higher variance, due to the latency-heterogeneity between runs. However, a notable difference is that COoL selection under *timing attacks* now behaves similarly to the other two attacked configurations. Deeper analysis shows that consumers do indeed send most requests to their closest providers, but network latencies are such that if the closest provider is malicious, the next closest honest provider is sometimes too far away to compensate: to yield fresh enough responses and, consequently, enough NBS-assets.

Finally, to answer *RQ3*, we have shown that in a topology with heterogeneous latencies between pairs of consumer-providers, the position of a consumer in the topology highly impacts their chance at discovering NBS-assets, whatever the selection policy. This is especially true for honest consumers, which, if they use COoL selection and their closest providers are *malicious*, may be subject to an attack δ_{att} up to the latency difference between these *malicious* providers and their next-closest *honest* provider. Should they use DeSearch-like selection instead, then malicious providers can attack them with an arbitrarily high δ_{att}, yielding even fewer dNBSa.

7 Related work

The problem tackled by Blind Slashing is at the crossroads of multiple research domains, which we present and compare to our work in this section.

7.1 Decentralized search mechanisms

There are still relatively few works on decentralized search over decentralized resources, compared to those on more centralized approaches. Moreover, their focus lies on the search itself and its properties (e.g., censorship- and bias-resistance), rather than on information front-running. DHT-based solutions like HypeerCube [**?**] and Ditto [**?**] do not guarantee result completeness but only a response from the

fastest node. Censorship is particularly possible in HypeerCube [?], where only one or a few nodes are responsible for specific keywords, and can choose to not respond to requests, or to delay them. TheGraph [?] is an indexing mechanism for Ethereum smart contracts, where resources are indexed based on their perceived value by curators. Again, completeness is not guaranteed. Finally, DeSearch [?] provides completeness and integrity of results using TEEs, but is not resilient as-is to information front-running, as search providers have the power over the network interfaces that connect the TEE to the outside world.

7.2 Transaction-front-running-resilient protocols

Some mechanisms deployed by protocols that protect against transaction front-running (TFR) in blockchain could be used against information front-running (IFR) in search systems, as both concepts are related: while the former aims to order a specific transaction before others, the latter delays some responses in favor of others. Note that while solutions against TFR bear similarities with IFR in that they also play on the ordering and timing of events, IFR remains a different problem from TFR: TFR solutions cannot be used as-is to protect against IFR. We discuss below the extent to which their mechanisms can be repurposed to protect against IFR.

Torres et al. [?] present the absence of transaction confidentiality and the blockchain miner's ability to choose transaction order as key causes for TFR. To this end, transaction-ordering systems were proposed [? ? ?], ensuring that individual servers cannot influence the ordering of transactions. Translating their protocols to our context, given such ordering mechanisms, existing acknowledgment messages sent back to the client can be repurposed as a response message to the client's request. The response time after computation will then be that of the fastest server in the network. If the closest servers are malicious, then the client can still suffer from timing attacks, as is the case with Blind Slashing. Nonetheless, requiring consensus for a request-response scheme is computationally expensive (considering n nodes, there is at least $O(n^2)$ in message complexity with TOB [?]; or $O(n)$ optimistically with SNARK-THEMIS [?], but with added SNARK-proof computation time).

Fairy [?] also uses transaction sender anonymization through relays like TOR [?]. Because malicious consumers can still decide to communicate directly with providers, they can still be recognized by malicious providers. Meanwhile, honest responses are delayed through the high-latency layered network. As Section 6.3 shows, this creates an IFR disavantage for honest consumers by design, even without attacks.

The behavior of solutions presented above, which are originally tailored to protect against *transaction front-running*, is captured by the "multiprovider selection" class defined in Section 5. We have shown in Section 6.2.1 that IFR has a strong impact in multiprovider-selection-based solutions.

7.3 Oracle systems

Blockchains operate as closed systems: they must rely on other entities to access data that is located outside their storage structure. "Oracle systems" provide this dataflow between blockchains and the outside world. Request-response search systems on top of blockchains [? ? ?] can be seen as "pull-outbound oracles" [?]: they draw data from the blockchain and feed it to outside entities, in our case, the market consumers.

If we consider the potential usage of our contribution for inbound oracles, i.e., those that feed data to blockchains, the block generation period (e.g., around 12s in Ethereum 2.0 [?]) outweighs the subsecond gains from latency optimization. Additionally, there is no obvious translation of inter-consumer competition in that context, so no IFR and therefore, no need to protect against it. This is why our proposed solution Blind Slashing is rather suited for outbound oracles, that is, to serve users or systems that are not blockchains. For more details, recent surveys explore existing works on blockchain oracle systems [? ? ?].

7.4 Decentralized QoS-based provider selection

Provider selection protocols are more in line with our work, as they indirectly tackle the IFR problem. However, they usually operate in a non-adversarial setting, where providers are assumed to be honest and the only problem is to select the best one. In most cases, Blind Slashing uses similar mechanisms to select providers, but adapts them to the adversarial setting with a trusted actor which is the TEE. A close work with ours is Go-with-the-winner [?], where providers are selected based on their end-to-end latency, in a non-adversarial setting, without insights from the providers themselves. We augment this work to a setting with malicious actors: we leverage TEEs notably to protect against content attacks, and use COoL selection against timing attacks. sPOT [?], Spatial Power of Two, is based on a well-known mechanism called Power of Two Choices [?], where a client selects two providers at random and chooses the least loaded one. In the case of sPOT, the two providers are the closest ones to the client. We show in Section 6.2.2 that this protocol is subject to IFR attacks. In Blind Slashing, we use historic latencies instead of network distances to select providers. Other works, like DONAR [?], use an additional role in the protocol, a mapper or broker, to select providers. This adds communication costs that can be bypassed by malicious consumers, increasing by design their IFR advantage.

8 Conclusion

In this paper, we highlight how providers of a decentralized marketplace's search service are able to give favorable treatment to part of the userbase, through *information front-running* attacks, even if Trusted Execution Environments

(TEEs) are used to guarantee the integrity and confidentiality of the search mechanism.

We show that quality-of-service-aware provider selection as performed by our proposed mechanism Blind Slashing, on top of TEEs, is paramount to counter such malicious provider behaviour. Thanks to Blind Slashing, we show that whether users collude or not with malicious providers, they do not get an edge over others in terms of fresher market state information, as long as there is enough honest computing power to serve all requests, in networks with low latency heterogeneity. Future research directions include investigating the dynamic spawning or retirement of service providers over the network topology, tailored to the users' latency requirements to deal with the intrinsic heterogeneity of client latency distributions.

References