SpotProxy: Rediscovering the Cloud for Censorship Circumvention

Patrick Tser Jern Kon, *University of Michigan;* Sina Kamali, *University of Waterloo;* Jinyu Pei, *Rice University;* Diogo Barradas, *University of Waterloo;* Ang Chen, *University of Michigan;* Micah Sherr, *Georgetown University;* Moti Yung, *Google and Columbia University*

https://www.usenix.org/conference/usenixsecurity24/presentation/kon

This paper is included in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Proceedings of the 33rd USENIX Security Symposium is sponsored by USENIX.

SpotProxy: Rediscovering the Cloud for Censorship Circumvention

Patrick Tser Jern Kon Sina Kamali[‡] Jinyu Pei[†]
Diogo Barradas[‡] Ang Chen Micah Sherr[⋆] Moti Yung[◇,○]
University of Michigan [‡]*University of Waterloo* [†]*Rice University*
[⋆]*Georgetown University* [○]*Columbia University* [◇]*Google*

Abstract

Censorship circumvention is often fueled by supporters out of goodwill. However, hosting circumvention proxies can be costly, especially when they are placed in the cloud. We argue for re-examining cloud features and leveraging them to achieve novel circumvention benefits, even though these features are not explicitly engineered for censorship circumvention. SpotProxy is inspired by Spot VMs—cloud instances backed with excess resources, sold at a fraction of the cost of regular instances, that can be taken away at a moment's notice if higher-paying requests arrive. We observe that for circumvention proxies, Spot VMs not only translate to cost savings, but also create a high churn rate since proxies are constantly re-spawned at different IP addresses—making them more difficult for a censor to enumerate and block. SpotProxy pushes this observation to the extreme and designs a circumvention infrastructure that constantly searches for cheaper VMs and refreshes the fleet for anti-blocking, for spot and regular VMs alike. We adapt Wireguard and Snowflake for use with SpotProxy, and demonstrate that our active migration mechanism allows clients to seamlessly move between proxies without degrading their performance or disrupting existing connections. We show that SpotProxy leads to significant cost savings, and that SpotProxy's rejuvenation mechanism enables proxies to be replenished frequently with new addresses.

1 Introduction

Censorship circumvention is often fueled by supporters out of goodwill [93]. Whether they set up proxying infrastructure at their own cost [47, 56, 69, 74, 94, 110, 111] or rely on donations [26, 60], their available budget is eclipsed by the resources of nation-state censors. To sustain the circumvention ecosystem, therefore, censorship evasion apparatuses must be able to operate on constrained budgets and make effective use of every dollar contributed towards circumvention.

Today, proxying infrastructure is hosted in an array of network locations, including content distribution networks [54, 107], Internet service providers [103], residential homes [42], edge networks [76], and, perhaps unsurprisingly, *the cloud*. Despite the transformative impact of cloud computing on many workloads, we argue that *circumvention proxies* must use the cloud in a very different way to reap benefits. Our goal in this paper is to re-examine and fully realize the power of the cloud for hosting circumvention proxies in virtual machines (VMs), maximizing cost savings and censorship resistance.

On the one hand, many circumvention proxies already hail from the cloud. This includes many Tor relay nodes [30, 31], standalone Snowflake proxies [27], and VPNs (e.g., Psiphon [25] and Lantern [20]). The conveniences of cloud computing, including its pay-per-use model and stable performance, apply to circumvention systems as well as traditional cloud workloads (e.g., key/value stores, web services). To host circumvention proxies, a user can rely on VMs in the cloud, without provisioning physical nodes in a dedicated place.

However, cloud VMs represent a substantially different computing environment, with many intriguing features not found elsewhere. Simply "lifting-and-shifting" a workload—any workload—into cloud VMs often falls short; one must bear in mind which cloud features are uniquely suited for that workload, in order to maximize benefits [100]. This is especially true for circumvention proxies, a non-conventional workload for which the cloud is not explicitly engineered.

The inspiration for this project came from a well-known cloud feature, *Spot VMs* [39,45,61], which at first glance seem unrelated to our goal, but a deeper look reveals two interesting properties under the lens of censorship circumvention. Spot VMs are a special class of cloud instances, spawned by cloud providers just-in-time out of excess resources to maximize profit, and then taken back at a moment's notice when higher-paying tenant requests arrive. Compared to regular VMs, spot instances are sold at a much cheaper price, even when they encapsulate the same amount of compute, memory, and network resources. This aligns with the goal of *lowering the cost of hosting circumvention proxies* as much as possible. On the flip side, cloud providers may preempt Spot VMs any time and repackage the underlying resources as regular VMs for a higher price. This instability of spot instances is commonly viewed as a nuisance, since it incurs an undesirable burden for

most workloads. The tenant needs to acquire a replacement VM, slot it into their fleet with a new random IP address, and resurrect any interrupted workloads. For censorship evasion, however, we observe that this high churn is in fact *a powerful antidote against censor blocking*. Proxy instances are constantly reborn at different IP addresses, even though their previous IPs may have been blocked.

Motivated by these observations, SpotProxy is a censorship resistance infrastructure that maximizes *circumvention utility* as the first-order objective using cloud-native features. Despite its name, SpotProxy is not restricted to using spot instances. Rather, it may use any available VMs found in the current cloud catalogue, spot or regular, as long as they are among the cheapest. Moreover, SpotProxy intentionally creates a high churn in its fleet. The Spot VMs on its fleet already naturally come and go, but in addition, SpotProxy also releases VMs as they age and spawn replacement VMs at new IP addresses to increase unblockability.

We envision that one way of using SpotProxy will be for a censorship circumvention sponsor (e.g., an organization like Tor, a VPN provider, or even a regular user) to deploy it in a public cloud (e.g., Amazon). They either take donations or pay out-of-pocket to create a fleet of SpotProxy instances, which in turn proxy communication from censored regions for whoever is in need. With SpotProxy, sponsors and contributors know that their resources are put to the most effective use, explicitly optimized for advancing the circumvention endeavor. Censored clients obtain entry to the SpotProxy fleet using conventional methods (e.g., email, moat [101]), but once they are connected, their proxy instances are constantly shifted from one VM to another to defend against IP-based blocking and enumeration attacks [33, 50, 109]. In a similar spirit, the complex and often-changing cloud price structure also leads to periodic cost arbitrage and proxy migration to cheaper VMs—dovetailing efforts to reduce the cost of the infrastructure and increase its churn. Importantly, client connections remain stable, and their communication seamless, as SpotProxy takes special care to preserve ongoing flows even when proxies are migrated across VMs. At any given point in time, the SpotProxy sponsor only pays for a (potentially small) number of proxies; but in the aggregate, these proxies comprise an increasingly larger IP footprint as they hop across VMs, creating an untenable collateral damage if the censor blocks all of the enumerated/observed addresses.

At its core, SpotProxy has two key mechanisms: *infrastructure rejuvenation*, which keeps around a cheap and fresh VM fleet as its workforce, and *client migration*, which ensures seamless client connectivity despite constant movement.

The first component of SpotProxy, the *rejuvenator*, constructs the underlying VM infrastructure and keeps it cheap (for cost savings) and fresh (for anti-blocking). To reduce cost, SpotProxy dynamically assesses the costs and benefits involved in acquiring variously-configured VM instances. This involves navigating the pricing variabilities of Spot VMs in relation to regular VMs, across cloud zones and regions, with a particular attention to circumvention-specific features, such as multi-NIC instances (which can host multiple proxies thus reducing per-proxy cost), ingress/egress preferences (for serving different censored regions and destinations), and sometimes, the ability to swap in a new IP address without any other change to the VM. This real-time arbitrage leads to the decommissioning of existing VMs in preference of cheaper ones. This churn is further amplified by keeping the VMs fresh, so that their IP addresses only host proxies briefly before being reallocated to unrelated cloud services, increasing collateral damage for a blocking adversary. Each VM is attached with a rejuvenation timer, and upon firing, the rejuvenator works to find a replacement with similar or cheaper cost. Complete with the natural dynamism of Spot VMs in our fleet, the underlying infrastructure is kept in constant flux.

The second component of SpotProxy is the *relocator*, which migrates active clients from one proxy instance to another as instructed by the rejuvenator. Whether a relocation event is due to cost arbitrage, Spot VM reclamation, or rejuvenation, the relocator can actualize frequent changes to client-to-proxy assignments. This enables SpotProxy to proactively and quickly hand over clients while keeping their connections alive, thus allowing SpotProxy to exert arbitrary control over both fleet composition and updated assignments. This further involves maintaining the state of available proxy instances, client-to-proxy assignments, session states across proxies (e.g., per Turbo Tunnel [53]), and seamless session resumption upon client-to-proxy reassignment. Furthermore, the relocator mechanism is proxy-independent—the same design pattern applies to different proxy implementations in re-engineering them slightly to be relocatable.

Our current prototype is compatible with the deployment of two prominent proxying protocols over SpotProxy-operated cloud instances: Wireguard [48] and Snowflake [42]. A live deployment and evaluation of SpotProxy's performance revealed that the system is able to support active migration at scale (migrating thousands of clients and hundreds of proxies simultaneously) while meeting the reclamation deadlines of cloud providers by a wide margin. In addition, we experimentally verified that SpotProxy preserves the continuity of client connections across migration events, imposing a negligible throughput degradation. We also evaluated SpotProxy' long-term cost savings by resorting to historical AWS EC2 pricing data, and experimental deployments. Our results show that SpotProxy would have been able to achieve ∼90% instance cost savings (and ∼74% total cost savings when considering network costs[1]) across several months in operation, assuming a move from ad-hoc proxy deployments in on-demand cloud instances to deployments on multi-NIC Spot VMs acquired via SpotProxy's cost arbitrage mechanism. Finally, we evaluated the circumvention efficacy of SpotProxy against multi-

[1] Network egress costs can be significant too. We analyze this in §11.1.

ple censor behaviors based on Nasr et al.'s game-theoretical simulation framework [80]. By adopting the proxy assignment algorithm from the same framework, we found that SpotProxy can provide access to ∼60% of legitimate clients even when censors control half of the connected clients (i.e., sybils); though we note that these are results obtained from a generic censor model not specialized to SpotProxy. These results support our belief that SpotProxy will drastically fuel circumvention as a next generation, cloud-based technique.

2 Why Rediscover the Cloud?

Cloud computing is hardly a new concept—more than 94% of enterprises already use the cloud [1]. In public cloud providers, compute, memory, and network resources are packaged as "virtual machines" (VMs), sold at varying prices in a pay-as-you-go model. Many vendors are vying for the market, ranging from large-scale service providers such as Amazon, Google, Azure, Digital Ocean, and Oracle, to smaller players such as OVH [85]. The cloud has in turn transformed how modern workloads are hosted, optimized, and served [79].

Owing to the success of cloud computing, it is not uncommon to host circumvention proxies in cloud VMs today [25, 27, 30, 31]. Some proxies even provide cloud configuration scripts [30, 31] for automated deployment. However, today's proxies use the cloud in a "lift-and-shift" manner, from residential homes/enterprises to virtual machines, with little customization for the circumvention endeavor. While this is sufficient for leveraging the more common features (e.g., payment model, ease of use, robust performance), it leaves vast opportunities largely untapped. The non-profit nature of censorship circumvention also means that the rising cloud costs that led to myriad business repatriation [16] will eventually impact proxy sustainability as well.

2.1 Benefits of the cloud

The **benefits** of cloud VMs can be felt by many workloads, circumvention proxies included.

First, cloud providers strive to make virtual machines easy to deploy and use, and proxy developers often supply cloud configuration scripts for automation. This makes cloud-based proxies far easier to deploy [75] than alternative circumvention solutions that reside in core networks [69, 74, 111] or at the edge [42, 47, 76]. Second, high availability is characteristic of the cloud, and providers dedicate substantial resources to support "five or more nines" [3]. VM-backed proxies, therefore, stand in contrast to user- or web browser-based counterparts (e.g., Snowflake [42] and MassBrowser [81]), which run on less reliable residential broadband networks and get shut down without notice. Third, the cloud provides higher performance, with VM resources customizable over a range of product classes (e.g., dedicated, shared, or fractional). Thus, VM-backed proxies can handle much more traffic and connections [89], whereas proxies on user laptops compete for resources with other applications. To sum it up, the cloud offers a commercial grade solution to circumvention proxies as it does for other workloads.

2.2 Downsides of the cloud

The commercial nature of the cloud, however, is at odds with the non-profit nature of circumvention endeavors, leading to important **disadvantages** surrounding cloud expenses.

Cloud VMs are expensive, and they are charged even when the underlying resources are idle. For instance, if a proxy fleet uses ten VMs with 2 vCPUs and 8GB memory[2], it costs about $600 per month even if some proxies are underutilized or blocked. To provision VMs with more CPU, memory, disk, and network resources, or even public IP addresses, leads to additional cost. Cloud costs are also steadily increasing, and this has given rise to the ongoing trend of cloud repatriation [16], where tenants move their workloads out of the cloud back to on-prem infrastructures due to the steep expenses.

This is particularly problematic for circumvention proxies, both for proxy sponsors/contributors and censored clients. For the former, financial strain is not uncommon especially when dealing with usage surges [60]. For instance, Lantern recently resorted to accepting donations and utilizing credit cards to sustain its operations and maintain stable access while scaling up its services in response to surging user demand [2]. One could instead ask proxy users to pay, but this increases the barrier for censored clients to access the free Internet. Censored clients could be financially constrained and need donations from governments and NGO entities for circumvention [23, 26, 90]. Payment channels for proxies could be restricted [32]. The act of paying for VPN services could be criminalized [40]. Free VPNs are thus in frequent demand, as evidenced by the increased usage during free access campaigns by IVPN [90] and the predominance of free VPN services in Russia [32], despite concerns that these services have opaque ownership and have been known to harvest user data and sell personally identifiable information to data brokers [52]. Thus, for sustainable operation of VM-backed proxies, we must minimize the necessary expenditures. For proxy sponsors and contributors, we must ensure that funds maximally contribute to circumventing censorship. For censored clients, we must extend free service to the broadest user base possible to increase its overall impact and accessibility.

2.3 Unleashing the cloud's full potential

We believe that a promising path forward is to re-examine the specific requirements of circumvention proxies, and support these goals with an infrastructure that leverages unique features of the cloud. Spot VMs [38, 39, 45, 61] are poised to offer a steep cost reduction, ranging from a guaranteed 60% (GCP) up to 90% (AWS/Azure), a price competitive to even niche providers (e.g., OVH [85])[3]. Spot VMs are also conducive

[2] `m6a.large` AWS EC2 instance and `b2-7` OVH instance

[3] OVH's roadmap hints at a future Spot VM offering [84], highlighting their emerging importance among cloud features providers aim to implement.

to autoscaling, unlike reserved instances [36] which require upfront lump sum payments. Although Spot VMs can be reclaimed at a moment's notice [38], this naturally refreshes the fleet with new random IP addresses [86, 87]. SpotProxy pushes these ideas to the limit by embracing these properties.

SpotProxy also leverages a few other cloud-native features. Cloud VMs can be deployed in chosen compute regions, so that services can reach their clients in closer proximity. In our case, this works nicely for supporting different censored regions and censored content. Some VMs can be assigned with multiple virtual NIC with public IP addresses [6, 17, 21, 46]. While this feature is originally designed for multi-homing scenarios [9], under the lens of circumvention, it can potentially provide further cost saving. Censors enumerate proxies by their IP addresses, so we can host a different proxy on each virtual NIC. Elastic IP addresses are another feature, allowing a VM to disassociate from its IP address on-the-fly and acquire a new one. Last but not least, the cloud hosts a variety of services that may be critical for high-profile businesses operating within censored regions. Censors are faced with significant collateral damage should they blindly block communications towards public cloud IP addresses [119]. In sum, our central principle is to reconsider cloud features and translate them into novel, anti-censorship benefits, thereby unleashing the cloud's full potential as a robust and cost-effective circumvention infrastructure. We emphasize that SpotProxy is an initial step in this direction, with further exploration detailed in §11.

3 SpotProxy

SpotProxy aims to satisfy the following **design goals**:

(1) **Minimal cost.** Proxies deployed on cloud instances are expensive. SpotProxy aims to provide maximum cost savings, a prerogative for effective circumvention.

(2) **Cloud-native unblockability.** The cloud, collectively, offers near-limitless resources (e.g., instances, or public IP addresses). SpotProxy should take advantage of this to make blocking more challenging for censors.

(3) **Seamless client connectivity.** Pursuing our two aforementioned goals requires frequent reassignments of clients to proxy instances. Client connectivity and network performance should not be impacted as a result.

3.1 Threat model

We assume that SpotProxy clients are located within regions under the influence of a state-level adversary who can monitor and manipulate clients' traffic flowing within its jurisdiction. Our assumption centers around a rational adversary who is mindful of the potential collateral damage stemming from the use of indiscriminate, coarse-grained blocking policies. For instance, the adversary refrains from outright blocking large swaths of IP address spaces pertaining to cloud providers due to potential collateral damage, an assumption adopted by existing cloud circumvention tools (§10). In addition, the adversary operates within computational bounds, lacking the capabilities to compromise standard cryptographic assumptions (e.g., to decrypt clients' traffic or forge signatures).

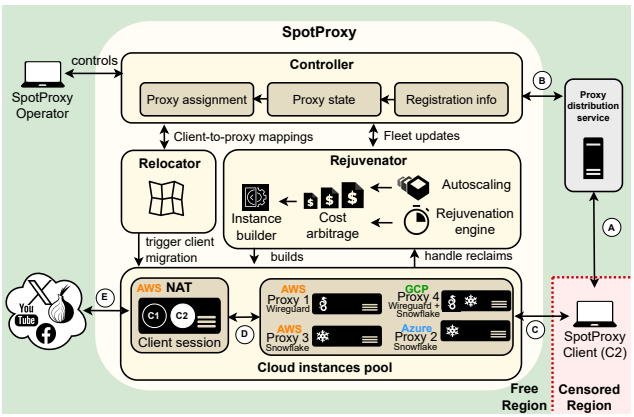


Figure 1: High-level view of SpotProxy's architecture.

However, the adversary has the power to deploy seemingly-legitimate SpotProxy clients to acquire proxy identifiers and engage in active probing. Given extensive research related to traffic analysis and active probing attacks [41, 57–59], we also assume that the proxying protocols that are deployed on top of SpotProxy's circumvention substrate already deploy adequate countermeasures, such as dedicated traffic obfuscation layers [112], probe-resistant proxy implementations [57, 59], or proxy-assignment algorithms [49, 80, 102].

3.2 Architectural overview

As shown in Fig. 1, the rejuvenator and the relocator components are our main innovation, but SpotProxy also features a central controller for orchestrating the relocation of proxy instances. Below, we describe clients' typical workflow when using SpotProxy, and then describe the operation of SpotProxy's backend components.

Client workflow. To use SpotProxy, clients leverage a modified client-side proxy binary to Ⓐ register through some existing proxy distribution service (e.g., a domain-fronted [54] broker, moat [101]), which forwards such requests to the Ⓑ *central SpotProxy controller*. Upon registration, the client is assigned a proxy (via some existing proxy-assignment algorithm) whose address is communicated back to the client by the broker, and the client can then Ⓒ initiate a connection to the proxy. Clients' requests transit through a NAT which preserves client session state Ⓓ while data is exchanged with the Internet services the client wishes to access Ⓔ.

SpotProxy backend. The fleet of proxies handled by SpotProxy is in constant flux, requiring frequent client-to-proxy re-assignments. These fleet changes are handled by the controller and do not require clients to go through re-registration. To this end, the controller continuously collects information about client registrations from the distribution service, and proxy state from periodic heartbeat signals sent by the proxies

themselves (e.g., containing CPU utilization, network transfer volumes, or the number of connected clients) to inform the remaining SpotProxy's components that we describe below.

The *rejuvenator* (details in §4) acquires cheap VM instances for cost savings, and keeps them fresh to increase unblockability. This satisfies our design goals (1) and (2). The *relocator* actuates re-assignment decisions made by the controller by migrating clients to different proxy instances for seamless connectivity (details in §5). The *proxy instances* can be of any type, as long as they can be re-engineered to become relocatable (see §6). This procedure incorporates active migration support, alongside our NAT device implementing TurboTunnel [53] (this device can be deployed on regular VMs and horizontally/vertically scaled according to client demand). This satisfies design goal (3) by providing a stable endpoint from the perspective of the destination—that is, the destination (e.g., YouTube) is unaware of the client's changing proxies because its TCP endpoint remains the (static) NAT host and its TCP connection is unaffected by client migration. Our migration technique is designed to be integrated into existing proxy implementations (as later showcased in §6).

4 The Rejuvenator

The rejuvenator minimizes cost by acquiring spot VMs or other VMs of the cheapest type, and it creates churn by refreshing its fleet periodically.

4.1 Cost arbitrage

SpotProxy opportunistically and continuously searches for the cheapest VM instances through cost arbitrage, as Spot VM pricing resembles a marketplace with variable prices [95] (across hundreds of instance types, availability zones, dozens of regions, and even potentially across clouds). This differs from existing works that (1) have a static assignment upon initialization and do not change until reclamation occurs [4, 114], (2) are focused on a single region and thus lose out on price competitiveness [4], or (3) choose a mixture of instances (e.g., more expensive regular instances) to reduce reclamation risk [28]. The ability to opportunistically select inexpensive instances is enabled by our active migration mechanism (§5) that allows for arbitrarily frequent reassignments of clients to the current cheapest proxy instances, effectively turning circumvention into a fault-tolerant application. It periodically constructs a preference catalog for autoscaling and cost arbitrage purposes. The catalog provides an up-to-date cost-ranked instance list (akin to SkyPilot [114] and OpenCost [24]) based on the number of proxies required, common filter attributes (vCPUs, RAM, bandwidth), and two circumvention-specific filter attributes:

(a) *Multi-NIC enabled.* This attribute leverages the principle that public IP addresses are the core units in circumvention, each serving as a unique proxy endpoint. Thus, it exposes another dimension for cost arbitrage, since larger but more expensive instances tend to be equipped with more virtual NICs (vNICs) as well, lowering the overall per-proxy cost. For instance, the AWS `c6in.large` instance has three vNICs, effectively reducing the cost of hosting a single proxy by two-thirds.

(b) *Egress region preferences.* This attribute is useful for proxy services (e.g., VPNs) that cater to clients with specific requirements (e.g., accessing geo-filtered services, or network performance needs).

The SpotProxy controller then relies on the catalog to select instances, and invokes cloud-level APIs to provision the underlying VM resources, forming a proxy fleet.

4.2 Infrastructure rejuvenation

Given the high cost of cloud VMs, SpotProxy seeks to ensure that cloud expenses are efficiently steered towards enabling censorship resistance. This requires solving the principal challenge of "disappearing" infrastructure in the circumvention context, which has two main root causes: (a) circumvention proxies have to contend with an adversarial censor that can block access to proxies; and (b) Spot VMs are prone to reclamation by cloud providers. Both causes are unpredictable in severity and frequency.

To address these challenges, SpotProxy introduces the concept of infrastructure rejuvenation, which harnesses the cloud's capability of on-demand provisioning of resources to continuously replenish disappearing infrastructure to maintain a stable supply of proxies. In doing so, SpotProxy can maintain a small (and thus inexpensive) proxy fleet footprint at any given moment, but their hosting VMs are constantly in flux, resulting in an infrastructure whose aggregate footprint is so large that the blocking cost is untenable. Rejuvenation employs a strategy of periodic proxy swapping based on the age of the proxies. This operates under the assumption that they will eventually be blocked or enumerated, and sometimes, the blocking events may not even be detectable by SpotProxy. This strategy is dynamic and can be as frequent as necessary (depending on the availability goals of the circumvention sponsor), thanks to the capabilities of active migration (§5) which guarantees seamless connectivity despite rejuvenation. SpotProxy provides two forms of rejuvenation: IP-based (§4.2.1) and instance-based (§4.2.2).

4.2.1 Live IP rejuvenation

Live IP rejuvenation is simple and fast, allowing for near-instantaneous rejuvenation. Conceptually, live IP rejuvenation relies on the principle that the public IP address of a cloud VM (i.e., the proxy) is the only resource that needs to be refreshed in order to counter IP-based blocking. The IP address refresh is performed on a "live" VM instance that does not need to be shutdown/restarted.

Live IP rejuvenation workflow. The workflow proceeds as follows. First, a rejuvenation period is associated with each VM within the fleet. These VMs will be assigned statically allocated public IPv4 addresses (known as elastic IPs in

AWS [6] or static IPs in Azure and GCP [17, 21]), instead of using their dynamic ephemeral IP addresses (which can only be refreshed via §4.2.2) that are allocated by default. Once the rejuvenation period (Tx) for a VM instance is reached, a new elastic IP address is allocated by the SpotProxy controller. Via the relocator (§5), the censored client receives this new address along with an associated shared secret and resumes proxying over this new address seamlessly. Prior to migration, IP remapping is triggered: the current attached elastic IP is disassociated from the existing VM and immediately deallocated; in this way, we do not have idle IP addresses which would incur additional charges [7, 10, 18]. The newly allocated IP is then associated with this VM, concluding the remapping. Finally, the associated proxy process on the VM is restarted before it resumes handling new user requests.

The cost of live IP rejuvenation. The primary advantage of utilizing live IP rejuvenation lies in its capacity to rejuvenate multiple IPs linked to multiple virtual NICs on a single VM, where each can host a proxy. This approach can significantly lower costs—for example, a single AWS `c6in.large` VM equipped with three vNICs can support three proxies, effectively reducing the hosting cost per proxy by two-thirds—in addition to the cost-saving measures previously discussed. The only additional cost incurred is the cost of acquiring additional public IPs (priced at $\$0.005/hour$ in AWS and GCP)[4], which can be billed at per-second granularities [7], just like VMs themselves [5, 19, 22]. Unfortunately, live IP rejuvenation can be cost-ineffective on Azure if Tx is low, as Azure charges IP address allocations at per-hour granularities [10] (i.e., partial hours are billed as full hours).

4.2.2 Instance rejuvenation

SpotProxy's second form of rejuvenation works by replacing the VM instances themselves. This is the most general form of rejuvenation, since every cloud provider will enable autoscaling of their VMs. Conceptually, instance rejuvenation mitigates blocking by regularly creating new VM instances that possess distinct IP addresses.

Instance rejuvenation workflow. The workflow proceeds as follows. First, a rejuvenation period is associated with each VM instance within the fleet. Note that these VMs will be using their dynamic ephemeral IP addresses that are allocated by default; the implication is that we do not support multi-NICs on a VM (since these can only be rejuvenated via §4.2.1). Once the rejuvenation period (Tx) for a VM is reached, it is substituted with a replacement instance. To allocate this instance, a request is sent to our opportunistic instance builder. Since spot VM fulfilment is not guaranteed by cloud providers (fulfilment can take minutes to hours [77, 88, 115]), this request also contains an expiration timer (Ex) that instructs the preference generator to timeout if an instance type takes more than Ex to be fulfilled, and try acquiring the next preferred instance type instead. Once the instance is activated, active migration ensures clients resume proxying seamlessly.

Embracing reclamation. In §2, we discussed the unreliability of spot VMs, noting their lack of guaranteed availability and thus the potential for spontaneous reclamation by cloud providers, usually with a short notice period (e.g., 30 seconds in GCP [62] and 2 minutes in AWS [38]). Traditional approaches to handling these reclamations view them negatively due to their disruptive impact on workloads, such as necessitating restarts from scratch or frequent, resource-intensive checkpoints [114]. In contrast, in our censorship circumvention setting, SpotProxy embraces reclamation by treating it as a special case of instance rejuvenation: SpotProxy performs instance rejuvenation spontaneously once a reclamation notice arrives rather than waiting for the next Tx. This approach involves re-provisioning a new VM instance to which impacted clients are then migrated, while ensuring seamless network continuity and minimal performance impact, as detailed in §5 and §7.1. Since reclamation occurs unpredictably and instance rejuvenation may not complete during the reclamation's short notification period, SpotProxy ensures disruption-free communication by either using a lower Ex or migrating clients temporarily to other instances.

5 The Relocator

The relocator actualizes client-to-proxy reassignment decisions made by the controller (Fig. 1) by moving clients to new proxies as the proxy fleet composition shifts, or as dictated by the assignment algorithm [49, 80, 102] used by the controller. We refer to this process as *active migration*. Reassigning clients to proxies is especially crucial because fleet composition will continuously fluctuate as SpotProxy strives to (1) maximize cost savings via runtime cost arbitrage and autoscaling, and (2) counter blocking and cloud-provider-induced reclamation through infrastructure rejuvenation (§4). Active migration needs to fulfill two design goals:

- **Seamless.** Migration should not cause client connectivity to be disrupted.
- **Automatic.** Reassignments should not require decision-making from external parties (e.g., the controller should not need to passively wait for clients to indicate that they would like to receive a proxy identifier).

Active migration mechanism. Active migration involves the following components: (1) the SpotProxy *controller* that maintains up-to-date databases regarding both the proxy instances available and current client-to-proxy assignments, performs assignment decisions based on an assignment algorithm, and initiates assignment decisions by coordinating with the relocator to construct relocation messages and; (2) *clients* running the client-side proxy binary that are connected to their assigned proxies; (3) *proxies* serving such connections that are managed by the controller; and one or more (4) *NAT* (network

[4] IP remappings are monetized on AWS ($0.10 per remap), but our experimental validation in §7.3 show that this does not apply to live IP rejuvenation, as IPs are never reassigned to another VM.

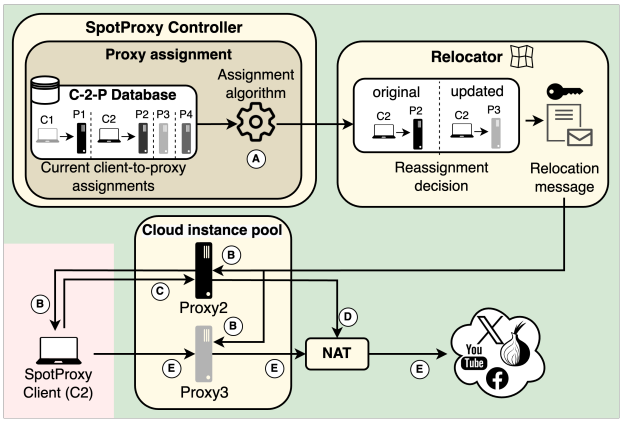


Figure 2: Active migration workflow. Re-assignment decisions made by the controller are actualized by the active migration mechanism, which allows the relocator to proactively communicate with both its managed proxy fleet (e.g., ACL) and clients (e.g., identifier and secrets) about its re-assignment decisions, and allows clients to instantly migrate to its newly assigned proxies while maintaining seamless connectivity.

address translation) devices that help maintain session state and constitute static endpoints to clients' destinations.

The workflow proceeds in phases, as illustrated in Fig. 2. We discuss these from the viewpoint of a single client, but in practice, migration would involve all clients assigned to the given proxy that needs to be migrated.

Active migration begins with the Ⓐ *planning phase* where the controller formulates a reassignment decision in response to a trigger (e.g., rejuvenation, reclamation). The controller consults up-to-date state such as the available proxy instances and current client-to-proxy assignments when forming reassignment decisions. Next, during the Ⓑ *notification phase*, the relocator informs the client of its impending reassignment via a relocation message that includes the identity and network address of the new proxy and any required secrets needed to establish connectivity with the new proxy. This message is securely relayed using the already-established channel between the client and the proxy. The controller's reassignment decision is also delivered to the new proxy, which inserts the client into its access control list (ACL).

Then, in the Ⓒ *confirmation phase*, the client parses the relocation request, and performs a three-way termination handshake with its current proxy. Next, in the Ⓓ *update phase*, the current proxy removes the client from its ACL, and informs the NAT to update its existing routes for the client. Finally, in the Ⓔ *resume phase*, the client authenticates and reconnects with the new proxy. Subsequent data packets are forwarded to the NAT (the egress point of SpotProxy traffic), which uses a Turbo Tunnel [53] design to identify the client's existing session to guarantee network continuity (e.g., without disconnecting existing TCP sessions to the client's previously-connected destinations).

Integrating relocation capabilities requires small modifications (see §6) to both the client- and proxy-side proxy software (e.g., Wireguard). The active migration process requires a client to have an ongoing connection with its current proxy to learn of its new assignment. If the client's proxy is suddenly blocked or otherwise becomes unavailable, then active migration is not possible and the client must re-register with SpotProxy as described in §3.2.

6 Implementation

We implemented a functional prototype of SpotProxy in approximately 5200 lines of code. We have open-sourced our code at https://github.com/spotproxy-project/spotproxy. SpotProxy currently supports operations on AWS (through `boto3` [37]), although its design applies to other clouds that offer programmable APIs for resource provisioning (e.g., GCP, Linode) [35, 63]. SpotProxy collects the fleet state (e.g., cost incurred by each VM in-use) by querying the appropriate cloud interfaces, and provisions/de-provisions VM instances based on various conditions that include cost arbitrage, autoscaling, and custom rejuvenation actions. The appropriate proxy implementations are then loaded/started within the VM.

The most significant modification that SpotProxy requires of existing proxies is that they need to be re-engineered to be relocatable. In what follows, we describe our implementations for two popular proxies, Wireguard and Snowflake.

6.1 Relocatable Wireguard

Wireguard is a popular network tunnel operating at layer 3, implemented as a kernel virtual network interface for Linux. The SpotProxy controller (Fig. 1) for Wireguard is written in Python using the Django [14] web framework, and the databases it maintains are managed with Django's internal ORM. It interacts with the instance manager sub-component through GET/POST HTTP endpoints.

We then implemented a management layer around the Wireguard [48] tunneling protocol, to be installed on both the proxy instances and the client. It is designed to handle multiple clients, creating individual Wireguard threads for each. Within these threads, a bidirectional connection is established linking the client, our management system, and the external NAT server. Upon instance initialization, our system, equipped with an Apache HTTPS server, starts monitoring for migration directives from the controller. Additionally, it incorporates a network traffic controller that manages data flow through the Wireguard tunnel. This controller reroutes outgoing data from the tunnel to the external NAT, and incoming data is similarly processed before entering the tunnel.

Upon a migration event, the management system receives Wireguard key configurations of clients (containing the client's public IP and Wireguard key) involved in the migration, and the traffic controller appends the migration header

(containing the replacement proxy's public IP and Wireguard key) to an arriving packet destined to the associated clients, which is encrypted by Wireguard, and delivered to the client. On the client side, the management system parses this migration header, updates the Wireguard configuration file, and restarts its Wireguard tunnel process, before reconnecting to the new proxy and resuming the network session with its last acknowledged sent packet.

6.2 Relocatable Snowflake

We made targeted modifications to Snowflake's codebase, particularly in the client, broker, and proxy components. We did not have to create a controller or NAT from scratch as with Wireguard since Snowflake's broker and server (that runs Turbo Tunnel) already provide these functions. We highlight the more prominent changes, starting with the planning phase:

When performing client-to-proxy assignments, the existing broker's knowledge of available proxies is restricted to proxies that are currently advertising their willingness to proxy client traffic (via an HTTP session opened with POST requests to the broker). This is sufficient for Snowflake browser-based proxies as they typically exist in large quantities, yielding a high probability that there is always at least one proxy advertising itself to the broker at any given time. However, this may not necessarily be the case for SpotProxy. To ensure we always have complete knowledge of available proxies, on the broker, we enhanced the `SnowflakeHeap` data structure to maintain an updated listing of proxy identifiers (e.g., IPs). Additionally, we introduced a `Client` struct that keeps a historical record of current assignments.

For the notification phase, we modified the broker's `/client` handler to issue periodic relocations. On the proxy, we set up an HTTP server with a `/transfer` handler capable of receiving relocation messages, which are then parsed and forwarded to the appropriate clients. On the client side, we added a `DecodeMessage` function to process the relocation messages and obtain details about the new proxy.

For the resume phase, on the client side, we implemented a `DirectConnect` function, enabling reconnection to a new proxy without broker involvement for seamless client handover. We also added a `/add` handler to the proxy's HTTP server to enable direct reception of SDP offers from clients, using the existing `makePeerConnectionFromOffer` function for client connection.

7 Performance and Cost Evaluation

We now evaluate SpotProxy in three dimensions: (1) how much load can SpotProxy active migration support, for varying client and proxy fleet sizes (§7.1)?, (2) how much cost savings can SpotProxy provide (§7.2)?, and (3) how fast can SpotProxy rejuvenate its infrastructure (§7.3)?. The efficacy of SpotProxy to evade censorship is evaluated later in §8.

Experimental setup. To evaluate SpotProxy performance and costs, we mostly rely on a *live network testbed*, built entirely on public AWS EC2 cloud VMs. Our controller (colocated with the rejuvenator and relocator) and NAT are both deployed on lightweight `m7a.large` EC2 instances (2vCPU and 8GB RAM), while our clients and network services acting as "free Internet destinations" are deployed on regular `c5.large` EC2 VMs (2vCPU and 4GB RAM) in various regions. For experiments whose results do not depend on SpotProxy's cost arbitrage (i.e., stress-testing the controller for active migration performance tests in §7.1), we also deploy proxies on regular `c5.large` EC2 VMs.

7.1 Active migration performance

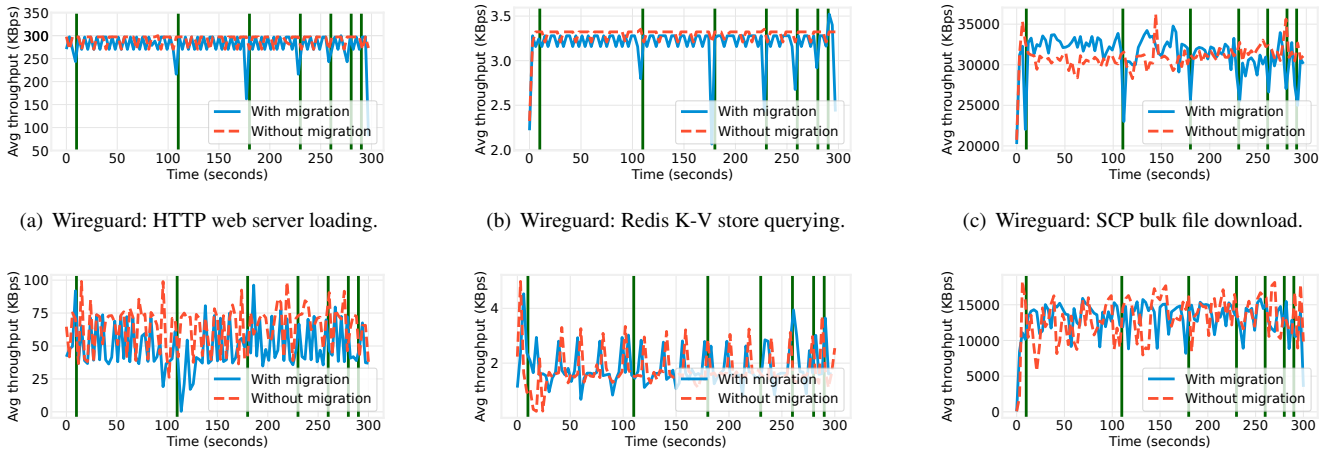
Active migration occurs when the controller decides that clients should be reassigned to different proxies due to cost arbitrage, autoscaling, rejuvenation, or reclamation. The main constituent costs of active migration are (1) the time required for the controller to perform reassignment decisions and notify all relevant parties, and (2) the time needed for clients to successfully reconnect to their newly assigned proxies. Next, we conduct stress tests to measure these overheads.

Migration efficiency. We assess the overhead of active migration by determining the degree to which it degrades the throughput of client connections. From the viewpoint of a single client, we evaluate our active migration proxy implementations across representative workloads, including (1) long-lived connections that outlive multiple migration events (i.e., an SCP bulk file download), and (2) short-lived connections that usually complete within a few seconds (i.e., accessing a web server, or a Redis K-V store).

As shown in Fig. 3, active migration incurs negligible throughput degradation (sub-second) from the viewpoint of our client, compared to a baseline in which no migration occurs. This holds for all our workloads, even as we gradually reduced the migration period (shown as green vertical lines) down to 5 seconds. Additionally, SpotProxy preserves connection continuity; for example, clients' SCP transfers were uninterrupted even across multiple migration events.

Migration scalability. We conduct scalability tests to ascertain SpotProxy's capacity to handle a large number of concurrent active migrations. Fig. 4 shows the average migration time for various configurations of clients and fleet sizes, for both SpotProxy-equipped Snowflake and Wireguard. A client's migration time is defined as the duration between the controller initiating the active migration event until the client connects to its newly assigned proxy. For each configuration, we perform three trials and plot the average. The variance across trials is small and the range of migration times is depicted by the (barely visible) error bars.

Our aim in the first two configurations (10 clients with 1 proxy vs. 100 clients with 1 proxy) is to stress a single proxy by increasing its load (i.e., the number of assigned clients) tenfold. Even with a heavy load, migration time is still minimal, requiring only 1.47s at worst.



(a) Wireguard: HTTP web server loading. (b) Wireguard: Redis K-V store querying. (c) Wireguard: SCP bulk file download.

(d) Snowflake: HTTP web server loading. (e) Snowflake: Redis K-V store querying. (f) Snowflake: SCP bulk file download.

Figure 3: Active migration incurs negligible client-side throughput degradation across diverse workloads and proxy implementations, even as migration frequency increases (green vertical lines), from a single client's viewpoint.

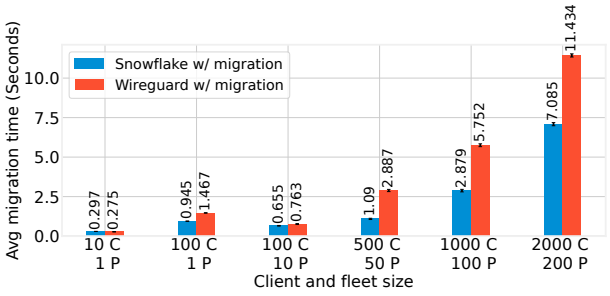


Figure 4: SpotProxy can handle a large number of concurrent active migrations. `100 C 10 P` represents a setup consisting of 100 clients and 10 proxies, where each proxy is assigned 10 clients (evenly-distributed). Error bars (barely visible) represent the range of values across all trials.

For the remaining four configurations, we focus on the *controller's* scalability by fixing a modest 10:1 client-to-proxy ratio while increasing the total number of proxies and clients (from 100 proxies and 10 clients, to 2000 clients and 200 proxies) that need to be migrated in parallel by the controller. Here, we notice a marked increase in average migration times, up to a maximum of 11.43 seconds for Wireguard. The higher times are largely due to communication overheads and our lack of parallelization—our current controller sends notification messages serially to proxies, which in turn also notify clients in a serial manner. While performance can clearly be improved by parallelizing such messaging, we note that the (unoptimized) migration times still support transparent (to the client) migration: they are well below the Spot VM reclamation notice of all cloud providers (the lowest is 30 seconds in GCP [62]), so we can be assured that all clients would have been migrated before reclamation takes effect. And, second, client throughput degradation is not noticeably worsened because the connection with the old proxy is terminated only after the new proxy becomes available.

7.2 Cost savings via cost arbitrage

To assess the potential cost savings SpotProxy can achieve through cost arbitrage, we analyzed historical spot VM pricing data, leveraging a comprehensive dataset [15] that encompasses AWS spot VM pricing histories over the last two years (Jun. 2022–Jan. 2024). Additionally, we cross-referenced this data with the AWS pricing history API (containing data for the past 90 days) to further validate it, and performed actual deployments (to confirm these cost-effective instances can be acquired). While our analysis focuses exclusively on AWS, due to the absence of comparable datasets for other providers, our findings remain relevant, since AWS's Spot VM pricing is competitive, even when compared to other major and specialized providers (e.g., Azure, GCP, OVH).

Fig. 5 depicts the analysis results for a few selected months of interest (a more comprehensive analysis over the last two years is shown in App. A.1). Our analysis covers regions within North America and targets non-burstable instances with at least 2 vCPUs and 4GB RAM. Using the historic pricing data, we derive the cost of operating a single proxy when using different strategies:

Static SpotVM denotes using the month's initially cheapest spot VM throughout, without applying cost arbitrage to adapt to price changes; while *Static on-demand VM* refers to the on-demand equivalent of the chosen static SpotVM instance. *Optimal single-NIC* refers to selecting an instance through cost arbitrage that exclusively employs its default NIC, whereas *Optimal multi-NIC* uses all the available NICs

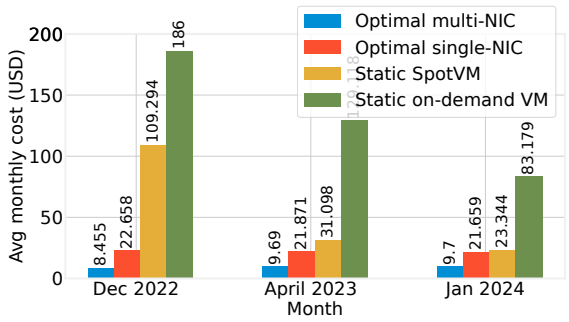


Figure 5: Average monthly cost of a single proxy. Cost arbitrage realizes significant cost savings that are stable and consistent (hovering at ∼$20 for single-NIC and ∼$9 for multi-NIC) by tempering the volatile costs of static Spot VMs.

(all with attached public IPs) on the same instances to potentially achieve further savings.

Fig. 5 shows that cost savings through arbitrage are stable and consistent, hovering around $20 for single-NIC and $9 for multi-NIC. Starting our analysis with Jan. 2024, transitioning from on-demand to Spot VMs yielded a 71% cost reduction. Implementing cost arbitrage added an extra ∼7.2% in savings, and adopting multi-NIC configurations further reduced costs by ∼55%. Initially, the most affordable spot VM, an `m7gd.large`, was priced at $0.0242 per hour, climbing to $0.0279 by month's end. Through cost arbitrage, which prompted 24 migrations within the month with intervals as brief as around 31 minutes and a median of about 10 hours, we shifted to more economical options such as the `m7g.large`, priced at $0.0241. Altogether, the shift from on-demand VMs to multi-NIC Spot VMs resulted in total savings of 88.3%.

Our analysis of the Apr. 2023 data saw even more significant savings from moving Static SpotVMs to those optimized via cost arbitrage, achieving savings of about 29.6% (single-NIC) and 68.8% (multi-NIC), respectively. This period saw cost arbitrage executed 98 times, with the quickest interval being roughly 30 minutes and a median time of 6 hours. Dec. 2022 showcased the most substantial savings with cost arbitrage activated only three times, but achieved savings of roughly 79.2% (Static SpotVM to Optimal single-NIC) and 92.2% (Static SpotVM to Optimal multi-NIC). Overall, moving from on-demand VMs to cost-arbitraged multi-NIC spot VMs yielded a maximum saving of 95.4%.

SpotProxy achieves these cost savings by facilitating arbitrarily frequent reassignments with minimal disruption (§7.1), enabling effective cost arbitrage, and the potential for further cost savings through autoscaling. This capability sets SpotProxy apart from existing systems (e.g., SkyPilot [114]) that offer cost arbitrage or autoscaling independently but lack seamless client reassignment.

7.3 Infrastructure rejuvenation stress tests

We next examine two questions: (1) how fast can we feasibly perform infrastructure rejuvenation? and (2) is rejuvenation cost sustainable? Our evaluation is motivated by two factors: Since Spot VM availability can be unpredictable, frequent rejuvenation might result in the acquisition of suboptimal (i.e., not the cheapest) VM types at a given point in time. Second, most cloud providers charge for IP address remaps [7], but the definition of remaps can be opaque (e.g., it is unclear whether AWS charges for associations/disassociations on a single VM). We therefore need to empirically validate if live IP rejuvenation results in additional remapping costs.

To answer these questions, we performed a series of experiments on AWS where we tested rejuvenation periods ranging from 2 hours to 2 minutes, with proxy fleet sizes ranging from 50 to 300. We performed three trials for each configuration, using non-burstable instances with at least 2 vCPUs and 4GB RAM within North American regions. Cost arbitrage did not occur during our experiments, and the cheapest instance was a `m6g.large` (with a maximum of three vNICs) at an hourly spot VM price of $0.0341 (excluding IP address costs). We stopped our trials at two minutes, as we observed it takes ∼10 seconds to associate/disassociate IPs (required by live IP rejuvenation), and ∼1.5 minutes to provision new instances[5].

Fig. 6 plots the cost of operating SpotProxy over a two-hour period, for various fleet sizes and proxy rejuvenation periods. As expected, operating more proxies incurs higher costs, and leveraging live IP rejuvenation and multi-NIC VMs can significantly reduce costs. Our results show that remapping costs do not apply to live IP rejuvenation (allowing us to freely associate IPs to all the available vNICs of an instance), while instance rejuvenation is generally able to acquire the cheapest (or within the top 5 cheapest) instance type even at large fleet sizes, making infrastructure rejuvenation cost sustainable.

Finally, during our evaluation, we acquired approximately 18,000 IPs which were all unique, over a period spanning multiple days. This finding is consistent with recent studies on large-scale allocations/deallocations [86, 87] that show the randomness of cloud IP assignments. SpotProxy could still reprovision another IP in edge cases where an IP is reassigned.

8 Circumvention efficacy

In this section, we evaluate the effectiveness of SpotProxy rejuvenation and relocation strategies in facilitating circumvention efforts against two main types of censor attacks.

First, we examine censors that aim to enumerate proxies and perform *IP-based blocking* (e.g., drawing comparisons with scenarios documented in previous studies [109] which

[5] A 90s provisioning time is under the current 120s warning-to-reclamation time practiced by AWS [38]. However, provisioning times which are longer than cloud providers' warning-to-reclamation times (e.g., 90s provisioning vs. 30s notification in GCP [62]) could mean that SpotProxy may have to very briefly resort to surplus capacity in the existing fleet to proxy the traffic of clients whose spot VMs will be reclaimed.

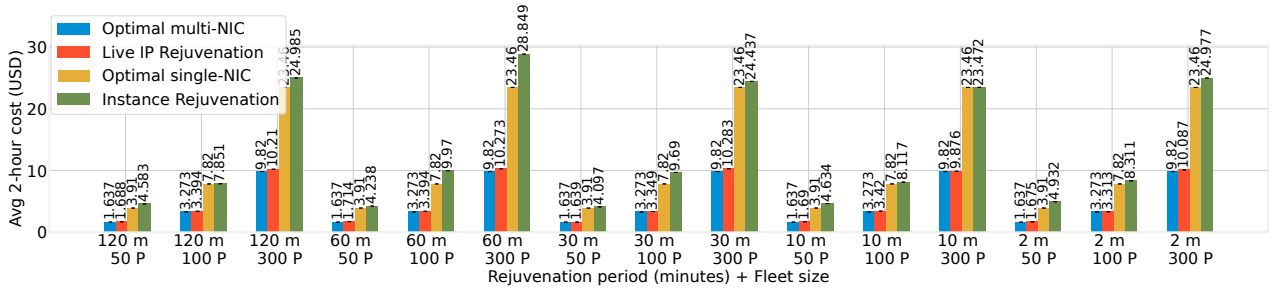


Figure 6: Both instance and live IP rejuvenation incur minimal cost overhead across rejuvenation periods and fleet sizes in AWS.

observed IP blocking in intervals as brief as 10 minutes, or only after a week [29]). We recall that SpotProxy does not consider traffic analysis and fingerprinting attacks within its threat model. Such attacks are deemed to be the responsibility of the underlying proxy technology.

Second, our analysis includes the threat posed by censors deploying *Sybil attacks*, where the adversary injects fake clients into the network to continuously acquire new SpotProxy proxy addresses to disrupt the system. This aspect of our evaluation aims to assess the resilience of SpotProxy's rejuvenation and relocation mechanisms against concerted efforts to undermine the system's circumvention efficacy.

Censor simulation platform. To test our system under a variety of scenarios, we employed the state-of-the-art game-theoretical framework introduced by Nasr et al. [80] (which we denote by *ENEM19*), also utilized by MassBrowser [81]; though we note that this is a generic framework not yet specialized to consider potential strategies by censors aware of SpotProxy. ENEM19 offers a comprehensive simulation environment, featuring over a dozen adjustable parameters. It provides a full-stack solution that includes a proposed *proxy assignment algorithm* designed to defend against Sybil attacks, alongside a detailed *censor model* that operates various strategies with different levels of blocking aggressiveness.

The ENEM19 simulator works as follows. First, its proxy assignment algorithm assigns a utility value to each client and proxy, where a high value represents a desirable proxy (e.g., alive for a long time) or trustworthy client (e.g., associated with few past blocked proxies). It then performs a greedy assignment by handing out the best proxies to the least suspicious clients. In turn, clients associated with blocked proxies have their client utility reduced, aiding in the differentiation between genuine users and those acting as censoring agents, with the latter eventually being excluded from the system.

Second, the censor controls a fleet of censoring agents (Sybils) posing as clients with the goal of acquiring proxies and blocking them via different strategies. We elaborate on two of the strategies introduced by ENEM19: (1) `ENEM19-aggressive` is a censor that commands its agents to block proxies as soon as they find them, and (2) the `ENEM19-optimal` censor behaves in a reactive manner to maximize its blocking power, by varying its waiting and blocking frequency/intensity, in an attempt to increase the number of proxies discovered and the total number of clients blocked. Censors will only gain partial knowledge of the proxy fleet, assuming some clients are benign. This is because ENEM19's assignment algorithm distributes proxies gradually rather than all at once, proxies are regularly rejuvenated, and Sybils associated with blocked proxies are systematically removed.

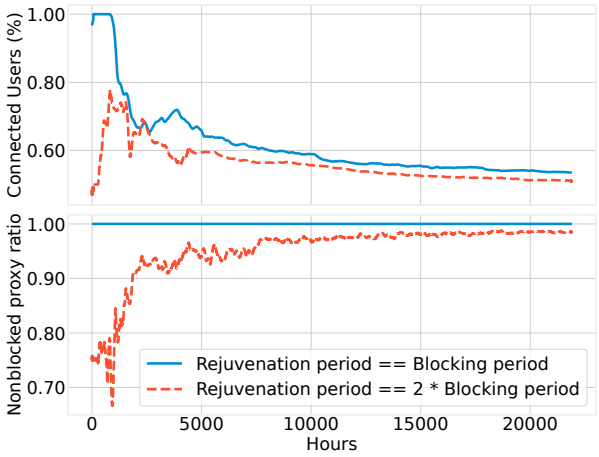


Figure 7: Results for `ENEM19-aggressive`. SpotProxy is capable of maintaining roughly 60% connectivity, despite censoring agents making up 50% of our client composition.

Simulator parameters. We largely adhered to the original choice of parameters used by ENEM19, except for its timescale: while ENEM19 assumes that operations occur at a day-level granularity, we instead increase the frequency of operations to a 2-hour granularity. This means that blocking/proxy assignment decisions occur every 2 hours. We also added a rejuvenation period to account for SpotProxy operations. We used two values of rejuvenation throughout our evaluation: 2 hours, matching the censor's blocking period; and 4 hours, two times slower than the censor's blocking period. The former ensures that SpotProxy infrastructure is rejuvenated as soon as the proxies have been blocked, while

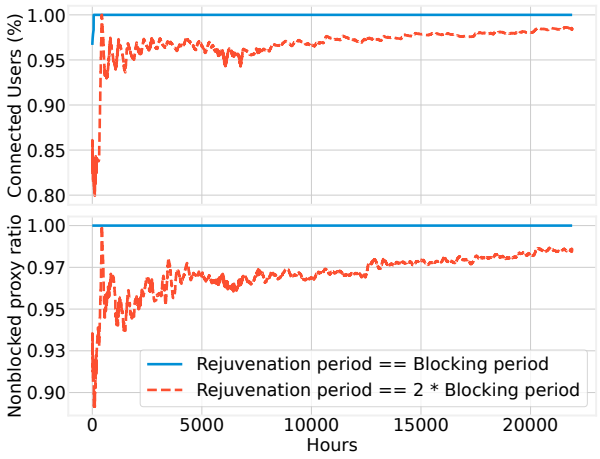


Figure 8: Results for `ENEM19-optimal`. SpotProxy is capable of maintaining over 90% connectivity, despite censoring agents making up 50% of our client composition.

the latter period only rejuvenates the infrastructure after some proxies have been blocked for 2 hours.

Simulation results. Although we tested multiple censor configurations, we highlight the more stringent censorship scenarios we handled in Fig. 7 (`ENEM-aggressive`) and Fig. 8 (`ENEM-optimal`), where censoring agents made up to 50% of our client composition. In our figures, *connected users* refer to the number of benign clients that remain connected, and *nonblocked proxy ratio* refers to the number of proxies that have not been blocked by censors. In App. A.2, we provide more details on less stringent scenarios.

First, we observe that censor blocking is largely ineffective in causing benign clients to be disconnected, for both models, since SpotProxy benefits from relocation; what actually causes clients to be disconnected is the fact that they end up being blamed by the ENEM19 assignment algorithm, which associates clients with too many blocked proxies over time. This also explains why out of all the models we tested, `ENEM-aggressive` was the most effective censorship model: since it performs blocking more frequently, it allows for this "implication-style" blocking to occur more successfully. This contrasts with the `ENEM-optimal` model, which prefers to wait longer before issuing client blocking decisions. However, when subject to the worst possible scenario, connectivity still hovers at around 60%, even though our setup was 12x more aggressive than `ENEM19`, with blocking occurring every 2 hours compared to their day-level granularity of blocking.

Second, we have a high nonblocked proxy ratio that hovers at around 90%. This highlights a caveat of `ENEM19`, in that it assumes clients blocked by implication will be removed from the system and never be able to acquire new instances; but this is a pessimistic assumption that may not be necessarily true in practice: SpotProxy could run a registration service to re-filter clients so that they can get back into the system despite having been blocked in the past, thereby increasing the connected ratio. We believe this to be a feasible effort since SpotProxy exhibits a high unblocked proxy ratio, meaning that SpotProxy maintains a large pool of available proxies to which re-filtered clients could get access.

9 Security discussion

This section discusses potential attacks to SpotProxy as well as attacks affecting proxy-based systems in general.

Financial DDoS. A censor could launch a large number of sybil clients to connect to SpotProxy, exploiting the system's autoscaling functionality to arbitrarily increase the costs of a SpotProxy deployment. However, standard sybil mitigations [49, 80, 102] apply. As another mitigation to this attack, SpotProxy's cost arbitrage module can ensure that cloud instance costs do not surpass a certain threshold, thus explicitly limiting the system's autoscaling capacity. While this does not prevent a potential DDoS to a SpotProxy deployment, it can actively avoid financial harm to SpotProxy operators.

Blanket ban of cloud services. A censor could block connections to all cloud VMs controlled by a given set of cloud providers, e.g., by applying strict IP blocking policies to the address space of such providers. While it is difficult to quantify the collateral damage experienced by censors willing to fully block connections towards cloud services, existing reports suggest that such actions cause severe economic repercussions to censoring states [11–13]. We note that other censorship circumvention systems making use of cloud storage services [43], content-delivery networks [54, 68, 120], and edge networks [76] operate under similar assumptions.

Blocking via strict allowlists. An adversary could attempt to list the IPs associated with popular cloud services, and block clients' connections aimed at IPs that are not on the list. However, since maintaining up-to-date allowlists as new Internet services are constantly added or updated is a complex task [76], censors have been known to be averse to such strategies due to potential collateral damage of overblocking [76, 99]. This is true even in today's cloud circumvention landscape where proxies are identified through static IPs. SpotProxy's dynamic rejuvenation improves the status quo by leveraging the built-in randomness of cloud IP assignments.

Proxy enumeration and blocking. A censor could launch a vast number of Sybil clients that would attempt to register in SpotProxy and enumerate proxy addresses for subsequent blocking. By design, SpotProxy makes this kind of attack impractical, as cloud instances will be continuously rejuvenated within short time intervals and alter proxies' IP addresses. Our evaluation earlier in §8 confirms this.

Proxy-to-client connectivity check. Censors may attempt to enumerate proxies and take action over users located within the censored region and which are found to be currently connected to IP addresses known to serve SpotProxy proxies.

We note that such an attack is not specific to SpotProxy, but inherent to most proxy-based circumvention systems.

10 Related work

Cloud-based censorship circumvention. Prior work has advocated for deploying anonymizing relays on cloud hosting providers [73,98], while exit bridges [119], deployed on cloud VMs, help bypass server-side censorship in Tor. While designed to be ephemeral like SpotProxy proxies, they do not allow clients to bypass state-level censorship policies. CloudTransport [43] uses public cloud storage for covert data transfer, while Camouflage [116] leverages personal cloud storage services like Dropbox. However, their performance lags behind traditional proxy services. Lastly, CDN-based techniques include those that let clients access blocked domains while appearing to access innocuous ones [54, 107], and those that allow access to censored content cached on CDNs [68, 120]. Of these, only domain shadowing [107] is able to provide access to any publicly available content, like SpotProxy.

Moving target defenses. SpotProxy can be understood as a moving target defense [71,92], dynamically shifting its attack surface by renewing the IP addresses of proxies, making it harder for adversaries to block them. Previous solutions have also drawn from moving target defenses to combat censorship. For instance, Heydari et al. [67] leverage mobile IPv6 to shuffle the IP addresses of public web servers, while Kon et al. [76] make use of programmable switches to periodically shuffle the domain-to-IP mappings of public services hosted within edge networks. Another related concept for SpotProxy involves a mobile target adversary [83], which can target and corrupt different fractions of parties in a protocol, but not all simultaneously. In SpotProxy, this corresponds to a censor that can enumerate a fraction of the proxies at any given time. To counter such threats, previous work has proposed proactively secure protocols for creating mobile target defenses [44, 51], e.g., using proactive secret sharing to distribute data among multiple storage servers and periodically refreshing cryptographic shares. Similarly, SpotProxy rejuvenates its fleet of proxy instances proactively.

Enhancing privacy through connection migration. Several encrypted protocols (e.g., QUIC [55], and Mosh [108]) allow clients to maintain session continuity despite IP address changes. SpotProxy however, enhances existing proxy implementations with active migration that allows for the seamless remote notification of updated peer addresses to connect to, while retaining session continuity through Turbo Tunnel [53].

Existing work has used connection migration for privacy-preserving communication. CoMPS [105] splits client traffic across network paths and protocols that support connection migration, making it harder for network adversaries to observe and analyze all traffic. MIMIQ [64] and RAVEN [78] use QUIC's connection migration to rotate client IP addresses and obscure the source of network flows. In contrast, SpotProxy uses connection migration to move clients across constantly changing proxies, not to prevent traffic analysis.

Cost reductions on the cloud. Before SpotProxy, researchers have already explored cost savings in the cloud (e.g., by profiling workloads and finding cost-effective VM-to-workload assignments [113]). Other works have used Spot VMs (either exclusively [65, 104] or mixed with on-demand VMs [66, 72, 117]) to lower cloud costs while increasing operational effectiveness. Further solutions for optimizing Spot VMs' cost, based on cost arbitrage and cost indexes such as HotSpot [96] and the work of Shastri et al. [97], respectively, migrate workloads to cheaper spot VMs while being optimized towards lowering reclamations and favouring applications that cannot tolerate frequent migration. In turn, SpotProxy is designed not only to embrace, but to aggressively trigger reclamations which are handled through the relocator. Skyplane [70] proposed network cost arbitrage, a strategy compatible with SpotProxy and which we aim to explore in future work, alongside other compelling directions (see §11 for more details).

11 Future work

SpotProxy offers many avenues for expansion, including:

11.1 Support for network cost savings

While we acknowledge the significance of network costs, our focus initially centers on the costs associated with instances, leaving network-related expenses for future exploration. This prioritization stems from the considerable weight that instance costs hold in cloud environments. For illustrative purposes, consider transitioning the entire Snowflake browser-based proxy ecosystem that is deployed on end-user devices (e.g., laptops), to AWS EC2 instances instead. Snowflake's network, as detailed in a recent report [42] by its creators, comprises approximately 130,000 proxies, and transfers around 30 TB of circumvention traffic per day. Rather than mirroring the one-to-one client-proxy ratio of the native Snowflake setup with 130,000 AWS instances, we make a conservative estimate that each EC2 instance could support 30 clients, given that cloud instances are more performant. Consequently, we assume that we only require provisioning 4,333 instances.

In our cost analysis, we standardize all calculations on a monthly basis to simplify comparisons. Regarding network costs [7], egress fees dominate, since (1) we can ensure minimal inter-VM costs between the proxy and NAT by situating them within the same region or zone, and (2) ingress costs are not a concern, as they are not charged. Applying AWS's tiered pricing model for egress traffic gives us a monthly network cost of $(0.09 \times 10 + 0.085 \times 40 + 0.07 \times 100 + (900 - 100 - 10 - 40) \times 0.05) \times 1000 = \$48,800$, for handling 900 TB of data transmission (30 TB per day).

For the instance cost analysis, we calculate expenses based on provisioning standard `m7g.large` instances with 2 vCPUs and 8 GB RAM. With an on-demand pricing rate of $0.077 per hour for the instance alone, the monthly cost for

operating 4,333 instances equates to $(0.077) \times 24 \times 30 \times 4333 = \$240,221.52$. Here, SpotProxy's use of multi-NIC `m7g.large` Spot VMs would instead cost $(0.0241) \times 24 \times 30 \times 4333 \div 3 = \$25,062.07$. Though we do not currently support network cost savings, this would still yield a ∼74% total savings of $240,221.52 - 25,062.07 = \$215,159.45$. This underscores the significance of both instance and network costs in our overall expense framework.

The figures presented serve merely as an example. To effectively manage network costs, one must recognize that the full integration of the Snowflake ecosystem into the cloud may not be essential or beneficial, especially since the value of existing browser-based proxies should not be overlooked, as they can continue to play a crucial role. Further, we can reserve cloud-based proxies for clients with lower bandwidth demands, and defer clients requiring substantial bandwidth allocations to proxy ecosystems where network cost are less of a concern (e.g., edge locations or user-hosted devices). Additionally, leveraging free tiers offered by cloud providers—e.g., AWS's 100 GB per month per account, and Oracle's 10 TB [82], among other offerings—can further mitigate costs.

11.2 Other directions

Integration with other clouds and proxies. We plan to support multi-cloud (e.g., GCP) and additional proxies (e.g., Tor native bridges, in addition to existing Snowflake support).

Cost calculation for burstable instances. This applies particularly to micro instances (e.g., EC2 `T4g` VMs), which operate on burstable performance models [8]. These models offer fixed or unlimited burst capabilities, potentially leading to unexpected costs when the allocated burst capacity is exceeded.

Performance variability among similar instances. We have assumed that instances with identical parameters (e.g., vCPU count) deliver equivalent performance. However, this may not always be the case, and different instances could exhibit varying performance levels (e.g., AWS instances, such as the `M7g`, that make use of the newer AWS `Graviton3` processors).

Integrating autoscaling with assignment. We incorporated basic autoscaling, such as scaling down when CPU utilization falls below 20%. However, integrating autoscaling more closely with the chosen assignment algorithm [102], currently not practiced, could improve cost efficiency or connectivity.

Cost savings via resource harvesting. Recent proposals in the cloud advocate for the harvesting of idle resources (e.g., CPU, storage) from VMs that have already been allocated to customers [34, 91, 106, 118]. While not yet made available by cloud providers, we expect that SpotProxy will also be able to utilize these to further reduce costs.

12 Conclusion

We have presented SpotProxy, a censorship circumvention infrastructure that hosts proxy instances in the public cloud while minimizing costs and improving unblockability. SpotProxy is in constant search for the cheapest virtual machines (VMs) available, reducing the cost of running circumvention proxies; at the same time, it also swaps older VMs out and spawns new ones at different IP addresses to increase unblockability. We adopted Wireguard and Snowflake for use with SpotProxy, and demonstrated the drastic cost benefits of SpotProxy, its resistance to blocking adversaries, and its ability to tolerate quick rejuvenations and relocations.

Acknowledgments

We thank the anonymous reviewers for their helpful comments and suggestions, and our anonymous shepherd for working with us to improve the paper. This work is partially funded by the Georgetown University Callahan Family Chair Fund, a VMware Early Career Faculty Grant, AWS Cloud Credit for Research, NSERC grant RGPIN-2023-03304, as well as NSF grants ITE-2226339, CNS-1942219, CNS-2106751, CNS-2107147, CNS-2214272.

References

[1] 26 Cloud Computing Statistics, Facts & Trends for 2023. https://www.cloudwards.net/cloud-computing-statistics.

[2] [Ars Technica] Fighting VPN criminalization should be Big Tech's top priority, activists say. https://arstechnica.com/tech-policy/2023/03/fighting-vpn-criminalization-should-be-big-techs-top-priority-activists-say/.

[3] [AWS] Achieving "five nines" in the cloud for justice and public safety. https://aws.amazon.com/blogs/publicsector/achieving-five-nines-cloud-justice-public-safety/.

[4] [AWS EC2] Allocation strategies for Spot Instances. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-fleet-allocation-strategy.html.

[5] [AWS EC2] Billing for interrupted Spot Instances. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/billing-for-interrupted-spot-instances.html.

[6] [AWS EC2] Elastic IP addresses. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/elastic-ip-addresses-eip.html.

[7] [AWS EC2] Elastic IP Addresses Pricing. https://aws.amazon.com/ec2/pricing/on-demand.

[8] [AWS EC2] Key concepts and definitions for burstable performance instances. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-credits-baseline-concepts.html.

[9] [AWS] Scenarios for network interfaces. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/scenarios-enis.html.

[10] [Azure] IP Addresses pricing. https://azure.microsoft.com/en-us/pricing/details/ip-addresses/#pricing.

[11] [Bloomberg] World's Worst Internet Clampdown Cost Myanmar $3 Billion in 2021. https://www.bloomberg.com/news/articles/2022-01-04/world-s-worst-internet-clampdown-cost-myanmar-3-billion-in-2021.

[12] [Carnegie Endowment] Government Internet Shutdowns Are Changing. How Should Citizens and Democracies Respond? https://carnegieendowment.org/2022/03/31/government-internet-shutdowns-are-changing.-how-should-citizens-and-democracies-respond-pub-86687.

[13] [Deloitte] The economic impact of disruptions to Internet connectivity. https://globalnetworkinitiative.org/wp-content/uploads/2016/10/GNI-The-Economic-Impact-of-Disruptions-to-Internet-Connectivity.pdf.
[14] [django] The Web framework for perfectionists with deadlines. https://github.com/django/django.
[15] [ericpauley] AWS Spot Price History. https://github.com/ericpauley/aws-spot-price-history.
[16] [Forbes] The Rise Of Cloud Repatriation: Why Companies Are Bringing Data In-House. https://t.ly/YR8Ik.
[17] [GCP Compute Engine] External IP addresses. https://cloud.google.com/compute/docs/ip-addresses.
[18] [GCP] External IP address pricing. https://cloud.google.com/vpc/network-pricing#ipaddress.
[19] [GCP] Spot VMs. https://cloud.google.com/spot-vms.
[20] Lantern. https://github.com/getlantern.
[21] [Microsoft Azure] Create a virtual machine with a static public IP address using the Azure portal. https://learn.microsoft.com/en-us/azure/virtual-network/ip-services/virtual-network-deploy-static-pip-arm-portal.
[22] [Microsoft Azure] Linux Virtual Machines Pricing. https://azure.microsoft.com/pricing/details/virtual-machines.
[23] [Open Technology Fund] Annual Report 2019/2020. https://public.opentech.fund/documents/OTF_Annual_Report_20192020_-_Final_v3.pdf.
[24] OpenCost: Cost monitoring for Kubernetes workloads and cloud costs. https://github.com/opencost/opencost.
[25] [Psiphon] Psiphon Circumvention System automation and server mangement. https://github.com/Psiphon-Inc/psiphon-automation/tree/master/Automation.
[26] [Reuters] Exclusive: U.S. targets Russia with tech to evade censorship of Ukraine news. https://www.reuters.com/world/exclusive-us-targets-russia-with-tech-evade-censorship-ukraine-news-2022-06-15/.
[27] [Snowflake] Help partner organizations to setup standalone snowflake proxies. https://gitlab.torproject.org/tpo/community/relays/-/issues/62.
[28] Spot by NetApp. https://spot.io/.
[29] [swgp-go: Userspace WireGuard Proxy with Minimal Overhead. https://github.com/net4people/bbs/issues/117.
[30] [Terraform registry] Tor (Google). https://registry.terraform.io/modules/Lirt/tor/google/latest.
[31] [tor-relays] Considering running a relay on AWS Free tier. https://forum.torproject.net/t/tor-relays-considering-running-a-relay-on-aws-free-tier/5722.
[32] [Wired] Russians need VPNs. The Kremlin hates them. https://www.wired.com/story/russia-vpns-internet-challenge/.
[33] How the great firewall of china is blocking tor. In *USENIX FOCI*, 2012.
[34] Pradeep Ambati, Íñigo Goiri, Felipe Frujeri, Alper Gun, Ke Wang, et al. Providing {SLOs} for {Resource-Harvesting}{VMs} in cloud platforms. In *OSDI*, 2020.
[35] Apache. Apache Libcloud. https://libcloud.apache.org/.
[36] AWS. Amazon EC2 Reserved Instances. https://aws.amazon.com/ec2/pricing/reserved-instances/.
[37] AWS. Boto3 documentation. https://boto3.amazonaws.com/v1/documentation/api/latest/index.html.
[38] AWS. Spot Instance interruption notices. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-instance-termination-notices.html.
[39] Azure. Use Azure Spot Virtual Machines. https://azure.microsoft.com/en-us/products/virtual-machines/spot.
[40] Ashley Belanger. Fighting VPN criminalization should be Big Tech's top priority, activists say. https://arstechnica.com/tech-policy/2023/03/fighting-vpn-criminalization-should-be-big-techs-top-priority-activists-say/.
[41] Benedikt Birtel and Christian Rossow. Slitheen++: Stealth TLS-based decoy routing. In *USENIX FOCI*, 2020.
[42] Cecylia Bocovich, Arlo Breault, David Fifield, Serene, and Xiaokang Wang. Snowflake, a censorship circumvention system using temporary WebRTC proxies. In *USENIX Security*, 2024.
[43] Chad Brubaker, Amir Houmansadr, and Vitaly Shmatikov. Cloudtransport: Using cloud storage for censorship-resistant networking. In *PoPETs*, 2014.
[44] Ran Canetti, Rosario Gennaro, Amir Herzberg, and Dalit Naor. Proactive security: Long-term protection against break-ins. *RSA Laboratories' CryptoBytes*, 3(1):1–8, 1997.
[45] Oracle Cloud. Preemptible Instances. https://docs.oracle.com/iaas/Content/Compute/Concepts/preemptible.htm.
[46] Oracle Cloud. Public IP Addresses. https://docs.oracle.com/en-us/iaas/Content/Network/Tasks/managingpublicIPs.htm.
[47] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *USENIX Security*, 2004.
[48] Jason A Donenfeld. Wireguard: next generation kernel network tunnel. In *NDSS*, pages 1–12, 2017.
[49] Frederick Douglas, Weiyang Pan Rorshach, Weiyang Pan, and Matthew Caesar. Salmon: Robust proxy distribution for censorship circumvention. *PoPETs*, 2016.
[50] Arun Dunna, Ciarán O'Brien, and Phillipa Gill. Analyzing china's blocking of unpublished Tor bridges. In *USENIX FOCI*, 2018.
[51] Karim Eldefrawy, Rafail Ostrovsky, and Moti Yung. *Theoretical Foundations for Mobile Target Defense: Proactive Secret Sharing and Secure Multiparty Computation*, pages 470–486. Springer International Publishing, 2018.
[52] Steven Feldstein. Government Internet Shutdowns Are Changing. How Should Citizens and Democracies Respond? https://carnegieendowment.org/2022/03/31/government-internet-shutdowns-are-changing.-how-should-citizens-and-democracies-respond-pub-86687.
[53] David Fifield. Turbo tunnel, a good way to design censorship circumvention protocols. In *USENIX FOCI*, 2020.
[54] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. Blocking-resistant communication through domain fronting. *PoPETs*, 2015.
[55] Internet Engineering Task Force. QUIC: A UDP-based multiplexed and secure transport. https://datatracker.ietf.org/doc/html/rfc9000.
[56] Sergey Frolov, Jack Wampler, Sze Chuen Tan, J Alex Halderman, Nikita Borisov, and Eric Wustrow. Conjure: Summoning proxies from unused address space. In *CCS*, 2019.
[57] Sergey Frolov, Jack Wampler, and Eric Wustrow. Detecting probe-resistant proxies. In *NDSS*, 2020.
[58] Sergey Frolov and Eric Wustrow. The use of TLS in Censorship Circumvention. In *NDSS*, 2019.

[59] Sergey Frolov and Eric Wustrow. HTTPT: A Probe-Resistant Proxy. In *USENIX FOCI*, 2020.

[60] Open Technology Fund. Surge and Sustain Fund. https://www.opentech.fund/funds/surge-and-sustain-fund/.

[61] GCP. Spot VMs. https://cloud.google.com/compute/docs/instances/spot.

[62] GCP. Termination and graceful shutdown of Spot VMs. https://cloud.google.com/kubernetes-engine/docs/concepts/spot-vms#termination-graceful-shutdown.

[63] Google. Google Cloud Python Client. https://github.com/googleapis/google-cloud-python?tab=readme-ov-file.

[64] Yashodhar Govil, Liang Wang, and Jennifer Rexford. MIMIQ: Masking IPs with migration in QUIC. In *USENIX FOCI*, 2020.

[65] Jashwant Raj Gunasekaran, Cyan Subhra Mishra, Prashanth Thinakaran, Bikash Sharma, Mahmut Taylan Kandemir, and Chita R Das. Cocktail: A multidimensional optimization for model serving in cloud. In *NSDI*, 2022.

[66] Aaron Harlap, Andrew Chung, Alexey Tumanov, Gregory R Ganger, and Phillip B Gibbons. Tributary: spot-dancing for elastic services with latency SLOs. In *USENIX ATC*, 2018.

[67] Vahid Heydari, Sun-il Kim, and Seong-Moo Yoo. Scalable anti-censorship framework using moving target defense for web servers. *IEEE Transactions on Information Forensics and Security*, 12(5):1113–1124, 2017.

[68] John Holowczak and Amir Houmansadr. CacheBrowser: Bypassing chinese censorship without proxies using cached content. In *CCS*, 2015.

[69] Amir Houmansadr, Giang TK Nguyen, Matthew Caesar, and Nikita Borisov. Cirripede: Circumvention infrastructure using router redirection with plausible deniability. In *CCS*, 2011.

[70] Paras Jain, Sam Kumar, Sarah Wooders, Shishir G Patil, Joseph E Gonzalez, and Ion Stoica. Skyplane: Optimizing Transfer Cost and Throughput Using Cloud-Aware Overlays. In *NSDI*, 2023.

[71] Sushil Jajodia, Anup K Ghosh, Vipin Swarup, Cliff Wang, and X Sean Wang. *Moving target defense: creating asymmetric uncertainty for cyber threats*, volume 54. Springer Science & Business Media, 2011.

[72] Pedro Joaquim, Manuel Bravo, Luís Rodrigues, and Miguel Matos. Hourglass: Leveraging transient resources for time-constrained graph processing in the cloud. In *EuroSys*, 2019.

[73] Nicholas Jones, Matvey Arye, Jacopo Cesareo, and Michael J Freedman. Hiding amongst the clouds: A proposal for cloud-based onion routing. In *USENIX FOCI*, 2011.

[74] Josh Karlin, Daniel Ellard, Alden W Jackson, Christine E Jones, Greg Lauer, David P Mankins, and W Timothy Strayer. Decoy routing: Toward unblockable internet communication. In *USENIX FOCI*, 2011.

[75] Seongmin Kim, Juhyeng Han, Jaehyeong Ha, Taesoo Kim, and Dongsu Han. Enhancing security and privacy of tor's ecosystem by using trusted execution environments. In *NSDI*, 2017.

[76] Patrick Tser Jern Kon, Aniket Gattani, Dhiraj Saharia, Tianyu Cao, Diogo Barradas, Ang Chen, Micah Sherr, and Benjamin E Ujcich. Netshuffle: Circumventing censorship with shuffle proxies at the edge. In *IEEE SP*, 2024.

[77] Sungjae Lee, Jaeil Hwang, and Kyungyong Lee. Spotlake: Diverse spot instance dataset archive service. In *IISWC*, 2022.

[78] Prateek Mittal Liang Wang, Hyojoon Kim and Jennifer Rexford. RAVEN: Stateless Rapid IP Address Variation for Enterprise Networks. *PoPETs*, 2023.

[79] Sean Marston, Zhi Li, Subhajyoti Bandyopadhyay, Juheng Zhang, and Anand Ghalsasi. Cloud computing—the business perspective. *Decision support systems*, 51(1):176–189, 2011.

[80] Milad Nasr, Sadegh Farhang, Amir Houmansadr, and Jens Grossklags. Enemy at the gateways: Censorship-resilient proxy distribution using game theory. In *NDSS*, 2019.

[81] Milad Nasr, Hadi Zolfaghari, Amir Houmansadr, and Amirhossein Ghafari. Massbrowser: Unblocking the censored web for the masses, by the masses. In *NDSS*, 2020.

[82] Oracle. Cloud Networking OCI Pricing. https://www.oracle.com/cloud/networking/pricing/.

[83] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks. In *PODC*, 1991.

[84] OVHcloud. [Github issue] Spot instances #196. https://github.com/ovh/public-cloud-roadmap/issues/196.

[85] OVHcloud. OVHcloud. https://us.ovhcloud.com/.

[86] Eric Pauley, Paul Barford, and Patrick McDaniel. {DScope}: A {Cloud-Native} internet telescope. In *USENIX Security*, 2023.

[87] Eric Pauley, Ryan Sheatsley, Blaine Hoak, Quinn Burke, Yohan Beugin, and Patrick McDaniel. Measuring and mitigating the risk of ip reuse on public clouds. In *IEEE SP*, 2022.

[88] Thanh-Phuong Pham, Sasko Ristov, and Thomas Fahringer. Performance and behavior characterization of amazon ec2 spot instances. In *IEEE CLOUD*, 2018.

[89] The Tor Project. VPN Village 2022: Run Snowflake proxies and strengthen Internet Freedom. https://wiki.digitalrights.community/index.php?title=Run_Snowflake_Proxies_and_Strengthen_Internet_Freedom.

[90] Reethika Ramesh, Ram Sundara Raman, Apurva Virkud, Alexandra Dirksen, Armin Huremagic, David Fifield, Dirk Rodenburg, Rod Hynes, Doug Madory, et al. Network responses to russia's invasion of ukraine in 2022: a cautionary tale for internet freedom. In *USENIX Security*, 2023.

[91] Benjamin Reidys, Jinghan Sun, Anirudh Badam, Shadi Noghabi, and Jian Huang. {BlockFlex}: Enabling storage harvesting with {Software-Defined} flash in modern cloud platforms. In *OSDI*, 2022.

[92] Sailik Sengupta, Ankur Chowdhary, Abdulhakim Sabur, Adel Alshamrani, Dijiang Huang, and Subbarao Kambhampati. A survey of moving target defenses for network security. *IEEE Communications Surveys & Tutorials*, 22(3):1909–1941, 2020.

[93] Adrian Shahbaz, Allie Funk, and Kian Vesteinsson. Countering an Authoritarian Overhaul of the Internet. https://freedomhouse.org/report/freedom-net/2022/countering-authoritarian-overhaul-internet.

[94] Piyush Kumar Sharma, Devashish Gosain, Himanshu Sagar, Chaitanya Kumar, Aneesh Dogra, Vinayak Naik, HB Acharya, and Sambuddho Chakravarty. Siegebreaker: An sdn based practical decoy routing system. *PoPETs*, 2020.

[95] Prateek Sharma, Stephen Lee, Tian Guo, David Irwin, and Prashant Shenoy. Spotcheck: designing a derivative iaas cloud on the spot market. In *EuroSys*, 2015.

[96] Supreeth Shastri and David Irwin. Hotspot: automated server

hopping in cloud spot markets. In *SOCC*, 2017.

[97] Supreeth Shastri and David Irwin. Cloud index tracking: Enabling predictable costs in cloud spot markets. In *SOCC*, 2018.

[98] Ali Shoker. Tormass: Tor for the masses domestic and monetized anonymous communication. *Procedia Computer Science*, 181:1216–1224, 2021.

[99] Ram Sundara Raman, Prerana Shenoy, Katharina Kohls, and Roya Ensafi. Censored planet: An internet-wide, longitudinal censorship observatory. In *CCS*, 2020.

[100] Byung Chul Tak, Bhuvan Urgaonkar, and Anand Sivasubramaniam. To move or not to move: The economics of cloud computing. In *USENIX HotCloud*, 2011.

[101] Tor Project, Inc. Tor Manual: Bridges. https://tb-manual.torproject.org/bridges/#using-moat, 2022.

[102] Lindsey Tulloch and Ian Goldberg. Lox: Protecting the social graph in bridge distribution. *PoPETs*, 2023.

[103] Benjamin VanderSloot, Sergey Frolov, Jack Wampler, Sze Chuen Tan, Irv Simpson, Michalis Kallitsis, J. Alex Halderman, Nikita Borisov, and Eric Wustrow. Running refraction networking for real. *PoPETs*, 2020.

[104] Marcel Wagenländer, Luo Mai, Guo Li, and Peter Pietzuch. Spotnik: Designing distributed machine learning for transient cloud resources. In *USENIX HotCloud*, 2020.

[105] Mona Wang, Anunay Kulshrestha, Liang Wang, and Prateek Mittal. Leveraging strategic connection migration-powered traffic splitting for privacy. *PoPETs*, 2022.

[106] Yawen Wang, Kapil Arya, Marios Kogias, Manohar Vanga, Aditya Bhandari, Neeraja J Yadwadkar, Siddhartha Sen, Sameh Elnikety, Christos Kozyrakis, and Ricardo Bianchini. Smartharvest: Harvesting idle cpus safely and efficiently in the cloud. In *EuroSys*, 2021.

[107] Mingkui Wei. Domain shadowing: Leveraging content delivery networks for robust Blocking-Resistant communications. In *USENIX Security*, 2021.

[108] Keith Winstein and Hari Balakrishnan. Mosh: An interactive remote shell for mobile clients. In *USENIX ATC*, 2012.

[109] Mingshi Wu, Jackson Sippe, Danesh Sivakumar, Jack Burg, Peter Anderson, Xiaokang Wang, Kevin Bock, Amir Houmansadr, Dave Levin, and Eric Wustrow. How the great firewall of china detects and blocks fully encrypted traffic. In *USENIX Security*, 2023.

[110] Eric Wustrow, Colleen M Swanson, and J Alex Halderman. TapDance: End-to-Middle Anticensorship without Flow Blocking. In *USENIX Security*, 2014.

[111] Eric Wustrow, Scott Wolchok, Ian Goldberg, and J Alex Halderman. Telex: Anticensorship in the network infrastructure. In *USENIX Security*, 2011.

[112] Diwen Xue, Michalis Kallitsis, Amir Houmansadr, and Roya Ensafi. Fingerprinting Obfuscated Proxy Traffic with Encapsulated TLS Handshakes. In *USENIX Security*, 2024.

[113] Neeraja J Yadwadkar, Bharath Hariharan, Joseph E Gonzalez, Burton Smith, and Randy H Katz. Selecting the best vm across multiple public clouds: A data-driven performance modeling approach. In *SOCC*, 2017.

[114] Zongheng Yang, Zhanghao Wu, Michael Luo, Wei-Lin Chiang, Romil Bhardwaj, Woosuk Kwon, Siyuan Zhuang, Frank Sifei Luan, Gautam Mittal, Scott Shenker, et al. SkyPilot: An Intercloud Broker for Sky Computing. In *NSDI*, 2023.

[115] Xiaomeng Yi, Fangming Liu, Zongpeng Li, and Hai Jin. Flexible instance: Meeting deadlines of delay tolerant jobs in the cloud with dynamic pricing. In *ICDCS*. IEEE, 2016.

[116] Apostolis Zarras. Leveraging internet services to evade censorship. In *ISC*, 2016.

[117] Qizhen Zhang, Philip A Bernstein, Daniel S Berger, Badrish Chandramouli, Vincent Liu, and Boon Thau Loo. Compucache: Remote computable caching using spot vms. In *CIDR*, 2022.

[118] Yanqi Zhang, Íñigo Goiri, Gohar Irfan Chaudhry, Rodrigo Fonseca, Sameh Elnikety, Christina Delimitrou, and Ricardo Bianchini. Faster and cheaper serverless computing on harvested resources. In *SOSP*, 2021.

[119] Zhao Zhang, Tavish Vaidya, Kartik Subramanian, Wenchao Zhou, and Micah Sherr. Ephemeral exit bridges for tor. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 253–265. IEEE, 2020.

[120] Hadi Zolfaghari and Amir Houmansadr. Practical censorship evasion leveraging content delivery networks. In *CCS*, 2016.

A Extended evaluation

A.1 Cost analysis

We extend our cost analysis with the remaining historical data mentioned in §7.2, executed over regions within North America and targetting non-burstable instances with at least 2 vCPUs and 4GB RAM. As seen in Fig. 9 and Fig. 10, our analysis extends from June 2022 to January 2024. Next, we provide general statistics on our overall analysis across this entire period. Transitioning from static on-demand VMs to static Spot VMs yielded a median cost reduction of 71.6%, a mean of 71.3%, a min of 41.2%, and a max of 88.2%. Transitioning from static Spot VMs to optimal single-NIC VMs yielded additional savings with a median of 9.2%, a mean of 20.9%, a min of 0.03%, and a max of 79.2%. Transitioning from optimal single-NIC to optimal multi-NIC provided extra savings with a median of 57.1%, a mean of 58.5%, a min of 55.2%, and a max of 62.6%. Transitioning directly from static on-demand VMs to optimal single-NIC yielded savings with a median of 80.8%, a mean of 78.8%, a min of 62.6%, and a max of 88.3%. Overall, transitioning directly from static on-demand VMs to optimal multi-NIC yielded savings with a median of 91.6%, a mean of 91%, a min of 83.8%, and a max of 95.5%. Finally, for cost arbitrage intervals, we have a median interval of 7.5 hours, a mean of 17 hours, a min of 16 minutes, and a max of 24 days.

A.2 Circumvention efficacy extensions

In §8, we tested `ENEM-aggressive` and `ENEM-optimal` with censoring agents making up 50% of our client composition. Here, we include the remaining configurations that we tested against, namely: 10% of censoring agents and 5% of censoring agents, for `ENEM-aggressive` (Fig. 11 and Fig. 12) and `ENEM-optimal` (Fig. 13 and Fig. 14) respectively.

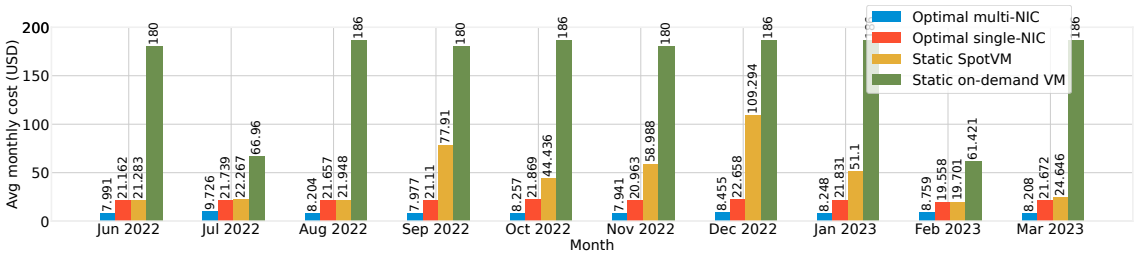


Figure 9: Average monthly cost of operating a single proxy, for months between June 2022 and March 2023.

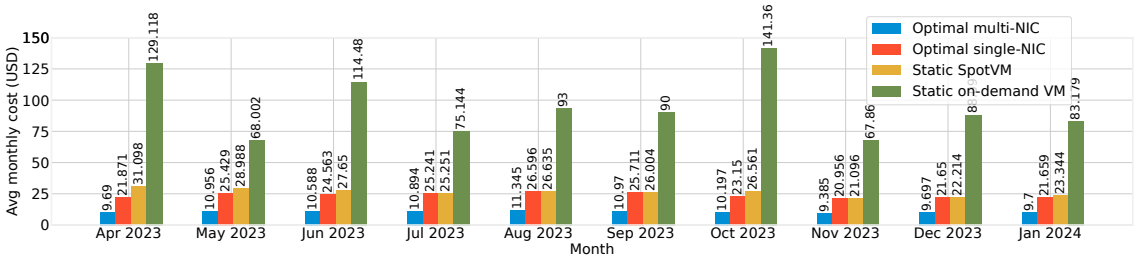


Figure 10: Average monthly cost of operating a single proxy, for months between April 2023 and January 2024.

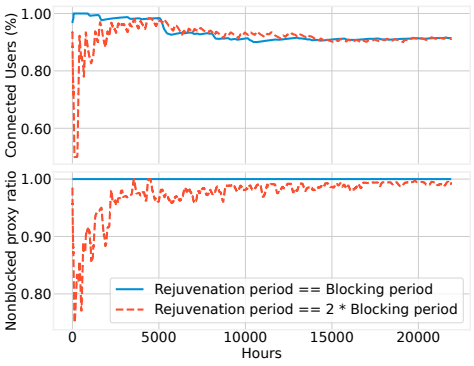


Figure 11: Results for `ENEM19-aggressive`. SpotProxy is capable of maintaining roughly 80% connectivity, with censoring agents made up 10% of our client composition.

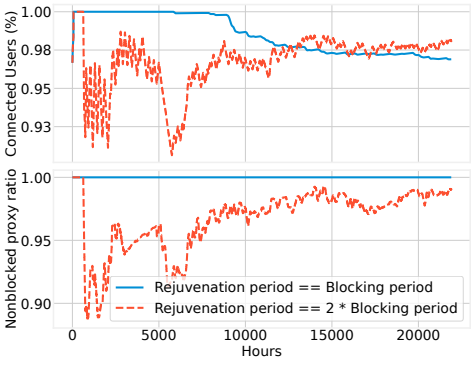


Figure 12: Results for `ENEM19-aggressive`. SpotProxy is generally capable of maintaining above 90% connectivity, when censoring agents made up 5% of our client composition.

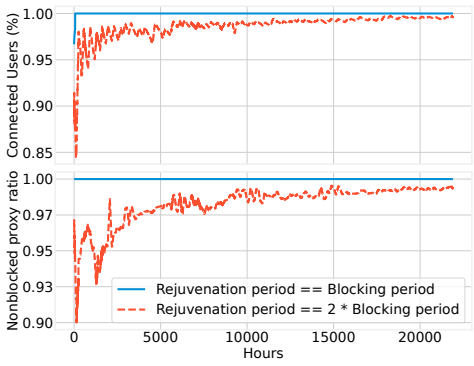


Figure 13: Results for `ENEM19-optimal`. SpotProxy is capable of maintaining generally above 90% connectivity, despite censoring agents made up 10% of our client composition.

Figure 14: Results for `ENEM19-optimal`. SpotProxy is capable of maintaining generally above 90% connectivity, when censoring agents made up 5% of our client composition.