CensorLess: Cost-Efficient Censorship Circumvention Through Serverless Cloud Functions

Dayeon Kang
University of Massachusetts Amherst
Amherst, Massachusetts, USA
dayeonkang@umass.edu

Jade Sheffey
University of Massachusetts Amherst
Amherst, Massachusetts, USA
jsheffey@cs.umass.edu

Mingshi Wu
GFW Report
gfw.report@protonmail.com

Pubali Datta
University of Massachusetts Amherst
Amherst, Massachusetts, USA
pdatta@umass.edu

Amir Houmansadr
University of Massachusetts Amherst
Amherst, Massachusetts, USA
amir@cs.umass.edu

Abstract

With the increase in Internet censorship globally, various circumvention tools have been designed and developed. However, the monetary cost of these tools deeply impacts both user choice and the sustainability of provider operations. Recent developments in censorship circumvention research attempted to achieve cost efficiency by utilizing Infrastructure-as-a-Service (IaaS) spot instances as bridges, but still incurred substantial expenses related to network connectivity and instance maintenance.

In this work, we present CensorLess, a circumvention proxy that leverages the unique benefits of serverless platforms. CensorLess comprises three components: a local proxy that manages client communication and enforces serverless security constraints, a function refresher that periodically regenerates bridges, and a live migration mechanism that maintains continuous connectivity. By design, CensorLess inherits key serverless properties, which are cost efficiency, ephemerality, scalability, concurrency, and high performance. Compared to existing low-cost, state-of-the-art circumvention techniques, CensorLess reduces operational costs by 97%, while simultaneously enabling robust censorship resistance by employing bridge rotation.

Keywords

Censorship Circumvention, Proxy Systems, Serverless Computing

1 Introduction

Internet censorship remains a significant barrier to global information access. To break this barrier, censorship circumvention tools have emerged as a solution. The monetary cost of censorship circumvention systems has been a critical factor both for users and system providers. For circumvention tool providers, securing funding is a common and big hurdle because maintaining a circumvention system includes the cost of renting servers, buying upstream bandwidth, continuously rotating IP addresses, and many other tasks [91, §5.3]. Open Technology Fund recently increased funding for circumvention tools due to a significant increase in the number of users of these systems [29, 48, 81]. The operational cost for large-scale circumvention tools can be prohibitively expensive [80]. For example, the Tor project spent $11,152.72 on a third-party server in February 2018 [69]. Recent surveys with users of circumvention tools [28, §4.3], [74, §5.2] show that price is almost always ranked in the top three most important requirements when users select a tool. Especially for users with limited-to-moderate technical expertise, price is a major factor in their decision-making; 71.1% (436 of 613) of users rank price in their top three [74].

Recently, proxy-based circumvention systems have started leveraging cloud computing to reduce operational costs. In proxy-based circumvention tools, a user in the censored region sends traffic to a proxy server that is allowed to access the Internet, and the proxy server routes the traffic to the true destination. The proxy server acts as a bridge or relay for the entire communication, enabling the user to access the Internet freely. The proxy can be positioned across various networks, including Internet Service Providers (ISPs)[40], Content Distribution Networks (CDNs) [96], edge networks [53, 64], and cloud infrastructures [17, 54]. Modern cloud-based circumvention proxies create distributed bridge networks with cloud instances that operate on usage-based pricing models to curb the costs [54].

In this paper, we leverage serverless cloud platforms (also known as Function-as-a-Service or FaaS) to reduce the operational costs of circumvention proxies compared to prior approaches. The serverless cloud supports request-based pay-as-you-go pricing, auto-scaling, high concurrency, and ephemerality. The per-request pricing model drastically reduces the cost in designing a circumvention proxy. More importantly, stateless short-lived serverless cloud functions maximize circumvention capability. When invoked, serverless functions create secure and isolated execution instances (e.g., containers) [12, 57]. These instances are ephemeral (e.g., maximum lifetime of 15 minutes in AWS Lambda [10]), and new instances are initiated with new IP addresses. This dynamic, randomized IP-address rotation [11] complicates IP-based blocking. Additionally, the auto-scaling and high concurrency inherently supported by serverless cloud enable CensorLess to scale to numerous clients effortlessly. We design CensorLess, the first serverless censorship-circumvention system that exploits these properties to deliver robust and extremely cost-efficient circumvention.

Designing a serverless circumvention proxy is challenging because the serverless cloud stack is largely opaque. We are limited

Proceedings on Privacy Enhancing Technologies 2026(3), 556–574

https://doi.org/10.56553/popets-2026-0096

in accessing information at layers below the application layer in serverless platforms, causing an inability to conduct socket communication or IP-layer datagram capturing. Disabled socket communication is particularly disruptive because it hinders the capability of the serverless function to act as a proxy. The alternative approach to network communication in serverless cloud requires the use of additional paid cloud services (e.g., API Gateway, Virtual Networks). However, incorporating additional cloud services is at odds with our goal of providing circumvention capability at minimal cost.

We design CensorLess to overcome the aforementioned challenges through three core mechanisms: *request translation*, *automatic bridge generation*, and *live migration*. Since a cheap circumvention tool is our core objective, CensorLess uses a local proxy server at the client side to avoid socket communication traffic. This proxy server translates Internet traffic into HTTPS requests, allowing users to browse websites transparently despite the limitations posed by serverless functions. To avoid DNS and TLS SNI-based blocking, CensorLess regularly retires serverless function bridges and creates new ones across multiple regions. To migrate connectivity between bridges and ensure uninterrupted service, CensorLess seamlessly transfers active network connections, communicating migration information through the current bridge to avoid exposing the new serverless bridge. By rotating bridges and capitalizing on the ephemeral nature of serverless functions through these three mechanisms, CensorLess enhances resilience against blocking or filtering by censoring agents.

Additionally, we make the observation that some serverless environments (e.g., AWS Lambda) provide the capability of domain-fronting [38] letting HTTPS requests present an allowed hostname while actually reaching a different backend. We hypothesize this capability persists to allow customers to pair serverless functions with CDN services such as Amazon CloudFront for operational convenience [50]. Because serverless invocations already originate from ephemeral, widely distributed IPs and can be reached via CDN edge domains, domain fronting lets clients present a legitimate-looking TLS SNI while the serverless bridge receives the real request — effectively camouflaging circumvention traffic among benign cloud flows. This design increases the collateral cost of censorship significantly, improves resilience by decoupling the visible and backend hostnames, and avoids additional long-lived infrastructure. CensorLess optionally leverages domain fronting in serverless deployments where the local proxy chooses the TLS SNI that appears legitimate and forwards traffic to the serverless bridge, which performs the backend routing.

CensorLess also supports a more stringent privacy-preserving mode that adopts a Virtual Private Server (VPS) to offer an encrypted communication channel and compatibility with SOCKS proxy, adding minimal cost to vanilla CensorLess. The private mode has a 20% cost increase over the vanilla system when it supports 50 serverless proxies. In Section 6, our evaluation demonstrates CensorLess's unparalleled cost-efficiency and performance. Compared to state-of-the-art circumvention proxies, CensorLess achieves at least 97% cost savings (63.3% cost savings for the private mode) and maintains costs below $3.50 per day, even when scaling to 300 proxies. Based on the findings from the experiments in Section 6, we provide operational guidelines for maintaining minimal costs when deploying serverless bridges. The contributions of CensorLess are summarized as follows:

- We develop CensorLess[1], a censorship circumvention proxy that maximizes cost savings without additional overhead, utilizing serverless computing design primitives.
- We implemented CensorLess on top of AWS Lambda, one of the most popular serverless cloud services, enabling support for other serverless clouds with minimal modifications.
- We thoroughly evaluate CensorLess's performance, censorship circumvention effectiveness, and associated costs using standard benchmarks, comparing it with state-of-the-art approaches. CensorLess reduces costs by at most 97% compared to the prior least-cost approach, SpotProxy [54].

2 Background and Related Work

2.1 Internet Censorship

Censors around the world have employed a variety of techniques and devices to inspect and filter network traffic.

Website censorship. Censors around the world often block websites and Internet services with a combination of various censorship techniques, which include, but not limited to, IP address blocking [19, §4], DNS injection [4, 8, 26, 46], HTTP Host-based filtering [72], TLS (E)SNI-based filtering [14, 19, 45], QUIC traffic filtering [31], and traffic throttling [3, 93]. These website censorship techniques are operated by diverse governments, China [15, 34, 45, 94], Iran [13, 58], Russia [55, 66, 73, 92], India [52], Turkmenistan [65], etc. Beyond national enforcement, censorship has expanded to regional levels, with local governments implementing their own filtering systems [33, 94].

Proxy detection. More resourceful and capable censors, including China [2, 90], Iran, and Russia, also identify and block proxy protocols and endpoints, creating a cat-and-mouse game between censors and netizens [6]. The censors have documented to target fully encrypted proxies [2, 90] (e.g., Shadowsocks [21], VMess [22], and Outline [51]), and TLS-based proxies [75] (e.g., Trojan [82] and Tor [27, 32, 87–89]). While China, Iran, and Russia have all been employing passive traffic analysis-based censorship techniques to identify proxy protocol [2, 41, 90], China is the only known country that also develops active probing infrastructures and techniques to more accurately identify proxy severs [2, 5, 7, 42].

2.2 Censorship Circumvention

Censorship circumvention remains a critical challenge for Internet users in regions with restricted access. Over the years, various approaches have emerged to address this challenge, evolving alongside advancements in network technologies and censorship techniques. This section examines the landscape of censorship circumvention tools with different approaches.

Proxy-based approaches. Proxy-based circumvention tools have been widely used to counter censorship techniques. These systems place proxy servers [25, 47] in uncensored regions, allowing clients to access the Internet through these intermediaries. Although relatively accessible to non-technical users, conventional tools such

[1] Source code is available here: https://bitbucket.org/umass-lab/censorless/src/main/

as VPNs [35, 95] often feature static IP addresses that are easily identified and blocked by censors. Researchers have experimented with placing proxies in various network environments, such as Internet Service Providers (ISPs) [40], Content Distribution Networks (CDNs) [96], and edge networks [53, 64] to improve resilience against detection. More sophisticated approaches leverage peer-to-peer architectures [36], such as MassBrowser [64], which uses normal users (i.e., volunteers) as proxies, or Tor [23], which implements multi-hop routing to provide stronger anonymity through circuit-based routing over multiple nodes.

Cloud-based circumvention systems. Prior works have leveraged unique characteristics of cloud services to design proxies in circumvention tools. CloudTransport [17] pioneered the use of cloud infrastructure to resist censorship by tunneling network traffic through encrypted cloud storage services. This approach utilizes the feature that public cloud storage systems provide a popular encrypted medium accessible from both inside and outside censor-controlled networks, making it difficult for censors to distinguish circumvention traffic from legitimate storage access.

While CloudTransport focused on using cloud products as encryption tunnels, SpotProxy [54] directly employed IaaS Virtual Machines (VMs) as bridges. Kon et al. used spot instances as proxies and implemented a function rejuvenation component that repeatedly searches for cheaper instances, creates new VMs or changes IP addresses, and migrates proxies accordingly. To maintain continuous connections during these transitions, they adopted Network Address Translation (NAT) techniques that temporarily relay traffic while migration takes place. Although SpotProxy effectively reduces costs through using spot instances compared to always-on instances, it introduces considerable operational overheads attributed to repeatedly searching for cheaper instances, implementing two types of rejuvenation (IP and instance), maintaining NAT infrastructure, and enduring longer configuration times for new instances. CensorLess addresses these inefficiencies by leveraging serverless computing to reduce operational complexity.

HTTP proxy. HTTP proxies [86] represent one of the most widely deployed proxy types in circumvention systems due to their simplicity and compatibility with standard web traffic. These proxies operate at the application layer, relaying HTTP requests from clients to destination web servers and returning responses. Traditional HTTP proxies operate on dedicated servers with static IP addresses, making them vulnerable to discovery and blocking by censors. Modern circumvention systems have enhanced HTTP proxying [67, 68] by implementing features such as request encryption and traffic obfuscation to improve resilience against detection. HTTP or SOCKS proxies offer a particularly accessible approach, allowing users to access blocked content without installing specialized software by submitting URLs into the browser's settings window. Despite continuous countermeasures by censors, HTTP proxying remains fundamental to many circumvention systems due to its flexibility and relative simplicity of implementation.

CensorLess builds upon this proxy mechanism while addressing its limitations by adopting core features of the serverless cloud architecture, providing a more cost-effective and resilient approach to HTTP-based circumvention.

Domain fronting. Domain fronting [38] represents a sophisticated evolution of HTTP proxying that uses Content Delivery Networks (CDNs) to circumvent Internet censorship. This technique allows HTTPS requests to appear as if they are destined for permitted domains while actually accessing blocked content. The core mechanism of domain fronting relies on using different domain names across TLS and HTTP layers. Normally, the HTTPS `Host` header, DNS query, and TLS Server Name Indication (SNI) extension have the same domain name. In domain fronting, a connection uses identical domain names in both the DNS query and the SNI of the TLS layer while employing a different domain in the HTTP `Host` header. This strategic inconsistency prevents censors from identifying the actual destination, as they can only monitor the DNS request and TLS SNI extension but cannot inspect the hostname in the encrypted HTTP layer. Upon receiving the request, the frontend server uses the hostname from the HTTP `Host` header to properly route communications to the covert destination.

3 Why use Serverless Computing?

Serverless computing or Function-as-a-Service (FaaS) [62] represents a modern cloud computing paradigm characterized by event-driven request execution with infrastructure management abstracted from developers. Serverless cloud shifts resource management, load balancing, function deployment, and execution responsibilities from developers to cloud service providers [85]. This approach enables developers to deploy stateless functions (i.e., we call serverless functions, which are function-level scripts) that automatically scale based on request load. Unlike the traditional Infrastructure-as-a-Service (IaaS) model (e.g., Amazon EC2, Google Compute Engine, and Azure Virtual Machines), the user does not have to manage the underlying hardware and software stack, and works with a high-level abstraction. Major cloud providers, including Amazon AWS, Google Cloud, and Microsoft Azure, have embraced serverless architecture, offering pay-as-you-go pricing models that typically charge around $0.20 per million invocations, with variations based on memory allocation, storage requirements, and execution duration. The combination of resources managed by providers, fine-grained scaling, cost efficiency, and streamlined development has positioned serverless computing as an increasingly important paradigm in cloud computing. While previous projects have utilized serverless functions as proxies—ranging from simple implementations to research-oriented reverse proxies for machine learning applications [61]—the potential of serverless in building effective circumvention tools remains unexplored. The inherent characteristics of serverless cloud present both opportunities and challenges in designing censorship circumvention proxies. In this section, we discuss the opportunities and challenges of using serverless cloud functions in developing CensorLess an inexpensive censorship circumvention proxy.

3.1 Merits of Serverless Computing

The stateless nature of serverless functions means all resources —including IP addresses— are ephemeral by design. Serverless functions are accessed via URLs or REST-based API calls rather than direct IP addresses, and their underlying IP addresses rotate unpredictably [11]. This fundamentally complicates censors' efforts to track or block users based on IP addresses and reduces the risk of exposing sensitive client information in censored regions.

Another notable property of serverless functions is that they are event-driven. Unlike traditional IaaS clouds that charge based on the entire period of renting and running compute instances regardless of usage, serverless platforms incur costs only when functions are invoked and executed. This event-driven model eliminates the need to maintain continuously running instances or to invest effort in finding lower-cost instances. For users in censored regions and circumvention tool providers, for whom cost is a critical factor, this request-based pricing model presents a notable advantage over traditional proxy approaches.

Serverless platforms offer autoscaling that automatically adjusts to function request volumes, deploying event-triggered instances to handle concurrent requests. This capability enables flexible handling of usage spikes without manual intervention and reduces operational overhead for circumvention tool providers while ensuring consistent service availability during periods of high demand.

Cloud providers typically offer serverless functions across numerous global regions, enabling rapid deployment of proxies across diverse geographic locations. This distribution capability enhances resilience against regional blocking attempts and allows for strategic positioning of proxies to optimize performance for users in specific censored regions.

3.2 Design Challenges

While the ephemeral nature of serverless functions provides notable advantages for censorship resistance, it simultaneously imposes significant constraints on system development and design.

- The serverless software stack is largely opaque to tenants, restricting the developer's access within the function instance (e.g., containers). Consequently, network visibility is restricted at the application layer, preventing the use of socket connections, packet capturing, kernel-level operations, etc. While cloud providers offer additional services like API Gateways, static IP addresses, and virtual networks to address these limitations, adopting these services diminishes the cost advantages of serverless computing. Thus, the primary challenge in building CensorLess is implementing effective network communication using only application-layer libraries while maintaining cost efficiency, ultimately leading us to utilize HTTP GET and POST methods as detailed in Section 4.4.
- When a serverless function hasn't been recently invoked, the instances hosting that function are removed from the deployment environment, and the cloud platform may need to initialize its execution environment before processing new requests, introducing "cold start" latency [57]. This latency can impact user experience, particularly for time-sensitive browsing activities.
- Serverless platforms impose constraints on payload sizes that may impact proxy functionality. AWS Lambda limits payloads to 6 MB (extendable to 20 MB under certain conditions) [56], Google Cloud Functions caps uncompressed HTTP sizes at 10 MB (32 MB in the second generation) [44], and Microsoft Azure Functions, while technically unlimited in response size, has practical limits around 250 MB [43]. These constraints necessitate careful design considerations for handling larger web content. We implemented CensorLess in an AWS environment and used Node.js to extend the payload size limit to 20 MB.

We carefully address the aforementioned challenges through innovative design patterns in building CensorLess. To the best of our knowledge, CensorLess is the first censorship circumvention solution that uses the inherent ephemerality in serverless cloud to achieve cost-efficiency and resilience to censorship.

4 System Design

In this section, we describe the design of CensorLess, our serverless proxy for censorship bypass.

4.1 Design Goals

We designed CensorLess with the following goals:

- **Cost reduction:** CensorLess is designed to minimize costs by providing a proxy service using only serverless functions, without requiring additional paid services offered by cloud providers, such as API Gateway or Virtual Networks.
- **Censorship resistance:** To succeed as a censorship circumvention tool, CensorLess is designed to effectively resist traffic tracking by IP addresses and blocking via DNS and TLS SNI through bridge rotation.
- **Scalability:** CensorLess delivers robust concurrency support, enabling service expansion from individual users to large-scale organizations while maintaining consistent performance.

4.2 Threat Model

We assume that CensorLess's clients are located in censored regions, and the serverless function, which acts as a proxy, is placed in a region that has not been censored. As a trust assumption for other components in CensorLess, operators must not disclose client-bridge mappings or inspect/record traffic in vanilla mode, and VPSs used for private mode must forward traffic correctly without logging. The censorship authority observes Internet traffic from the client and can block or interfere with the traffic if the client attempts to access censored websites.

The censor possesses the capability to inspect, manipulate, and block network traffic flowing within its jurisdiction. We consider every agent in our system to be rational. The censor is external to the cloud provider and acts as a rational adversary who aims to minimize collateral damage when implementing blocking measures. For instance, the censor is unlikely to block entire cloud provider IP ranges or domain names due to the significant collateral damage such actions would cause to legitimate services. Censors can also act as clients of CensorLess in an attempt to identify bridges and disrupt the service. They may actively probe suspected bridge endpoints and attempt to enumerate individual serverless function instances.

We also assume that the cloud provider acts rationally [54, 64], meaning they are willing to serve users as long as doing so does not expose them to any risk. For example, the cloud service provider may disclose encrypted network communications under coarse-grained security policies. In the real world, there is a possibility that cloud providers may behave irrationally, such as colluding with censors due to legal or economic pressure. We have discussed this scenario in Section 7.

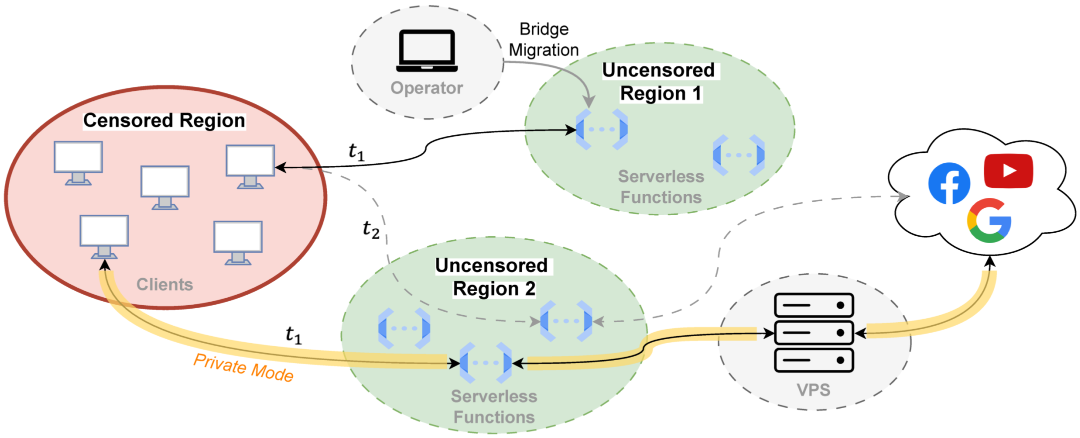


Figure 1: Overview of CensorLess operational workflow

We do not assume the censor has the ability to compromise end-user devices (e.g., by installing monitoring software), as this would undermine any privacy-enhancing technology. While censors might employ deep packet inspection (DPI) to identify traffic patterns associated with circumvention tools or block entire cloud providers such as Russia, the ephemeral nature of serverless bridges presents significant challenges to such identification efforts.

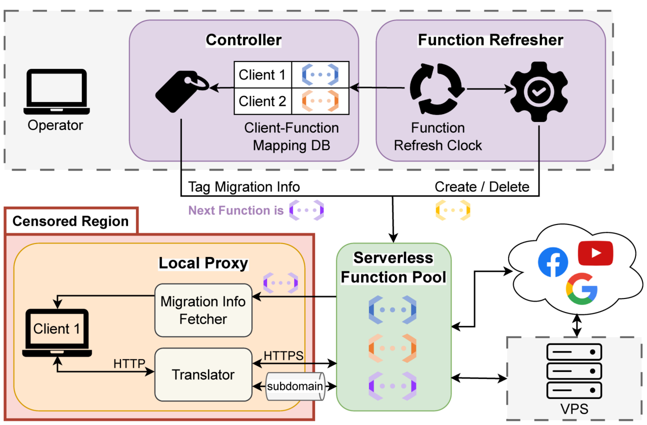


Figure 2: Overview of CensorLess architecture

4.3 Overview

CensorLess circumvents censorship by establishing ephemeral serverless computing-based bridges between clients in censored regions and their intended destinations. As Figure 1 illustrates, clients in a censored region send requests to serverless functions located in an uncensored region at time t_1. Serverless function bridges inspect the request and act as proxies, forwarding the traffic to the client's intended destination. The destination treats the serverless bridges as the actual clients and sends data back to them, then the function returns it to the original clients. The operator–any individual or organization willing to pay per-invocation costs–can be located in any region with cloud service access, managing serverless function bridges and orchestrating connectivity migration. When time reaches t_2 after a certain period, clients change their bridge to different serverless functions in random uncensored regions to continually obfuscate traffic patterns from censors.

CensorLess provides an additional component for privacy. The private mode operates with reduced cost efficiency while preserving the privacy of the client's requests. Based on the user's preference and their priority, the user can select between vanilla CensorLess and private mode. When the user selects the private mode, the client sends encrypted messages using a SOCKS proxy to a serverless function bridge over HTTPS. Then, the serverless bridge relays the message to the Virtual Private Server (VPS), and the VPS forwards the encrypted message to the destination on the internet. Except for sending encrypted messages and additionally connecting the VPS as a bridge to the Internet, the rest of the workflow is the same as the vanilla CensorLess, such as the operator's serverless bridge management and bridge rotation.

Taking a closer look at this system, it consists of three key mechanisms that addresses the key serverless-based design challenges, and four modules that support these key mechanisms. Key mechanisms are request translation, automatic bridge generation, and live connectivity migration. We will address these mechanisms in detail by grouping them into the client-side and the operator-side processes in Sections 4.4 and 4.5. The end-to-end workflow of the private mode is described in Section 4.6.

Our system comprises four primary modules as represented in Figure 2: controller, function refresher, serverless function pool, and local proxy. The *controller* and *function refresher* are managed by the operator and work together to orchestrate network connectivity migration between serverless bridges. Operating in coordination with the controller, the function refresher manages the lifecycle of serverless bridges. It periodically generates new serverless functions in batches across diverse regions, applies appropriate security configurations, and notifies the controller of their availability. It can adjust the scalability of the serverless function pool by defining the size of serverless bridges. The controller maintains a Client-Function mapping database that tracks serverless bridge assignments for each client. When the controller updates this database, it informs clients about their new bridge assignments by tagging information needed to migrate onto the current serverless function.

The *serverless function pool* contains batches of serverless bridges. Each bridge modifies incoming request headers from clients and composes new outgoing packets to appear as though the serverless bridge itself is the client. The private mode parses the HTTPS payload and forwards it as a full request. When the serverless bridge receives responses from the client's destination, it returns them to the client without modification. These serverless functions are lightweight (147 MB Docker image for vanilla system) as they serve only as minimal bridges.

The *local proxy* handles most request modifications to reduce processing overhead at the bridge. The biggest obstacle to the local proxy is the fundamental restrictions of FaaS. Since FaaS does not allow functions to communicate with a socket, the local proxy addresses these restrictions by establishing the application-level connections with the serverless bridge and adopting protocol conversion. The local proxy comprises two components: (1) a translator that converts client requests to HTTPS for serverless compatibility, using SOCKS proxy in private mode, and (2) a migration information fetcher that periodically detects and seamlessly transitions to new bridge URLs.

4.4 Client Side

A client starts CensorLess by installing the local proxy server, which is preconfigured with an initial serverless bridge URL. The serverless bridge is periodically updated in the background. After the client runs the local proxy, network traffic is relayed to the allocated serverless function bridge, and the intended data from the target destination is returned through the bridge. For transparent HTTPS communication with target websites, the local proxy server adopts a key mechanism, request translation.

Request translation. Request translation addresses the fundamental limitation of serverless functions: their inability to support socket-based communications, which are commonly used today. To overcome this constraint, CensorLess converts HTTP requests in the local proxy server using two principal components: a *translator* and a *migration information fetcher*.

The translator intercepts client HTTP GET and POST requests while listening on a designated port to avoid the HTTPS CONNECT method, which uses socket communication, and processes them for secure transmission to the serverless bridge. Given the security vulnerabilities inherent in standard HTTP requests, the translator converts them to HTTPS, modifying or removing irrelevant request headers in the process. The converted traffic is then forwarded directly to the assigned serverless function bridge through an HTTPS-encrypted tunnel. At the serverless bridge, minimal request modifications are performed to emulate an actual endpoint, reinitiate the HTTPS tunnel, and send traffic to the intended destination. Due to the inherent restrictions of serverless functions and the need to minimize costs without extra cloud services, the communication protocol is separated. When responses are received from the destination via the serverless bridge, the translator performs the reverse conversion, transforming HTTPS responses back into HTTP format for client consumption. This process includes removing unnecessary HTTPS-related headers and modifying response bodies when required. In private mode, a SOCKS proxy is used, and the encrypted message captured from the client is sent to the serverless bridge over an HTTPS tunnel by the translator. Additionally, the local proxy implements streaming responses and pipelining to enhance network performance.

To enhance resistance against detection by censors, the client exclusively communicates with its allocated serverless bridge rather than directly interacting with other system components. The migration information fetcher periodically checks for migration tags embedded in responses from the current bridge. When a new bridge URL is detected, it seamlessly migrates the connectivity without requiring external components or introducing communication overhead. This HTTP-based approach enables transparent and immediate migration for clients while maintaining continuous service.

4.5 Operator Side

To effectively obfuscate traffic patterns and enhance resistance against detection, CensorLess regularly rotates serverless bridges across various uncensored regions. The operator executes two key mechanisms to facilitate this rotation: automatic bridge generation and live migration. The system starts with a client registration to CensorLess through a local proxy server installation by a client. Upon registration, the operator randomly selects the next serverless bridge from the available pool, allocates it to the client, and records this assignment in the Client-Function mapping database. This initiates the ongoing cycle of bridge rotation and management.

Automatic bridge generation. The function refresher operates according to operator-defined time periods, creating new batches of serverless functions and removing those that have completed their operational cycle. When generating new serverless bridges, it employs a lightweight Docker image and applies strict security policies that limit access permissions to the minimum required for operation. Once new bridges are deployed, the function refresher notifies the controller via HTTP request, enabling subsequent client migration. After clients migrate away from bridges at the end of their cycle, the function refresher automatically removes these deprecated functions, ensuring that no bridge remains active long enough to be identified and targeted by censorship agents. The period until the bridge is identified in practice is expected to be 2 days, as reported by Fifield et al. [39]; Tor bridges in China are discovered within 2-36 days.

Live migration. The controller orchestrates migration by managing the Client-Function mapping database. Upon receiving notification of new bridge availability from the function refresher, the controller updates the mapping database, removing old bridge assignments and allocating new ones to clients. This update triggers the migration process. Under the stateless nature of serverless functions, the controller implements a tag-based migration mechanism. Rather than directly communicating migration instructions to clients, the controller attaches the URL of newly assigned bridges as tags. These tags are detected by the client's migration information fetcher, enabling seamless transition to the new bridge. The old bridge is automatically removed by the function refresher after clients complete their migration. Throughout this process, the controller maintains minimal communication with clients, primarily interacting with a serverless bridge. This design simplifies the migration process and reduces detectable communication patterns. The HTTP-based implementation enables this efficient live migration without requiring complex protocols or procedures. Through these coordinated client-side and operator-side mechanisms, CensorLess achieves continuous service while regularly altering network pathways, effectively circumventing SNI-based censorship.

4.6 CensorLess for Privacy

CensorLess provides public internet connectivity, circumventing censors, at minimal cost. However, a malicious cloud provider could potentially access data at the serverless bridge. Therefore, we present a private mode that preserves privacy for CensorLess, though it introduces a trade-off between cost/performance and privacy/anonymity. We compare the additional cost incurred by this trade-off in Section 6.3. While the privacy for clients against malicious operators at the serverless bridge is preserved, malicious operators at the VPS may analyze ingress and egress traffic patterns. We address this limitation in Section 7.

The privacy-preserving module deploys a Virtual Private Server (VPS) for the SOCKS protocol, long-lived connections, and message encryption, which increases operational costs. As shown in Figure 1, VPS is located after the serverless bridge and before the target destination in the uncensored region. Like the vanilla CensorLess, the connection between the client and the serverless bridge uses HTTPS, but the communication between the serverless bridge and the VPS server uses TCP.

Encryption model. To protect the data in the message via encryption, the client and the VPS server each have a public key and private key ($C_{pr}, C_{pub}, S_{pr}, S_{pub}$), and they share their public keys in the bootstrap step, coordinated by the operator. When a client sends a message to a server that contains an encrypted payload ($E(m)$) with the destination, the client encrypts the message with the server's public key ($c_{req} \leftarrow E(S_{pub}, E(m))$), and sends it to the server through the serverless bridge as an HTTPS payload. The server decrypts the message with its private key ($E(m) \leftarrow D(S_{pr}, c_{req})$) and validates the incrementing nonce included in the message. When the server sends the request to the target destination, the connection ID (*ConnID*) is granted by the server, and it is signed with the server's private key ($DS \leftarrow Sign(S_{pr}, ConnID)$). After the server receives the encrypted response ($E(r)$) from the target destination, the server attaches the signed connection ID to the encrypted response and encrypts the entire message with the client's public key ($c_{res} \leftarrow E(C_{pub}, DS|E(r))$). Finally, the client receives the response by decrypting the server's message using its private key ($E(r) \leftarrow D(C_{pr}, c_{res})$).

With this encryption model, four security properties are maintained. Since we use the Ed25519 [30] public-key signature system for encryption, the client and server are authenticated with Ed25519 public keys configured out-of-band. To prevent replay attacks, we adopt a monotonically increasing nonce. The signed connection ID proves that the client owns the connection, and all encrypted payloads secure confidentiality. A formal proof of the security properties of private mode is left for future work.

End-to-end communication. The client initiating the communication is a SOCKS proxy, and it forwards the message to the serverless bridge, embedding it into an HTTPS payload as a vanilla CensorLess to comply with the restrictions of the serverless function. The request (i.e., HTTPS payload sent by the client) includes the VPS server address in a SOCKS address format to make the connection with the server and the payload that is encrypted with the server's public key. When the serverless bridge receives this HTTPS request from the client, the bridge establishes a TCP connection with the VPS server, and it parses and forwards the HTTPS payload as a message. When the server receives the TCP segment from the bridge, it decrypts the payload using its private key and verifies the incrementing nonce for the public key it was given. Next, it either initiates TCP connections or ferries data between the client via the serverless bridge's connections or the target destination. The server buffers responses for each client that is identified by the public key until the client requests them through another serverless bridge invocation. Buffered responses are controlled and cleaned up with a timeout and a maximum buffer size.

When the server receives the response from the target destination, it encrypts the response using the client's public key and sends it back to the serverless bridge over the TCP connection. The serverless bridge places the TCP segment into an HTTPS payload and sends the request over HTTPS. Then, the client parses the payload and decrypts the actual response from the target destination with its private key. Finally, the client can route the data to the SOCKS communication channel.

4.7 Domain Fronting on CensorLess

To enhance its censorship circumvention capabilities, CensorLess can optionally leverage domain fronting supported by certain serverless platforms. It is a technique that conceals the true destination of HTTPS traffic by utilizing the discrepancy between different layers of communication protocols [38]. While major CDNs including AWS, Google, and Microsoft have withdrawn their support for domain fronting practices, largely to preserve their commercial relationships with censoring states [16, 37, 60, 77, 79], we discovered that domain fronting continues to be available for AWS Lambda. We tested the domain fronting on Microsoft Azure functions, but it does not allow a mismatched SNI and Host. By providing the additional option for connection establishment through domain fronting in CensorLess, we improve the censorship resistance of CensorLess by exploiting the fact that the censor has to pay a high cost to block serverless functions.

Mechanism. CensorLess adapts the traditional domain fronting for the serverless environment. In this approach, a client connects to a permitted cloud domain (e.g., `*.lambda-url.*.on.aws` for AWS Lambda) and sends an encrypted HTTPS request with a different `Host` header representing the censored destination.

This process involves (1) locally establishing the connection to the targeted destination domain, (2) setting the HTTP `Host` header within the encrypted channel to the specific serverless function bridge, and (3) utilizing the cloud provider's content delivery infrastructure to route the request appropriately. The local proxy server selects TLS SNI values to ensure that the domain fronting requests appear legitimate to both censorship systems and cloud provider infrastructure. This subdomain may be the same as in other serverless functions in the same region, even a completely fabricated subdomain unregistered with AWS.

This implementation is particularly effective because censors face a significant dilemma–blocking all traffic to major cloud providers would cause substantial collateral damage to legitimate services and applications, and the ephemeral serverless endpoints further complicate filtering efforts.

Enhanced resilience through serverless. Unlike traditional domain fronting implementations that rely on a limited set of front domains [38], CensorLess can distribute requests across numerous serverless functions deployed across different regions and namespaces within the cloud provider's domain space. This distribution significantly increases the cost of comprehensive blocking.

Even if AWS Lambda decides to disable domain fronting, CensorLess, without domain fronting, still maintains resilience against identification and filtering attempts by dynamically selecting the domains of serverless functions across multiple regions. Additionally, CensorLess allows operators to customize the domain selection interval, strengthening the system's robustness against censorship efforts. Incorporating domain fronting alongside CensorLess' censorship resistance mechanisms provides even greater blocking resistance by preventing actual front-end exposure and increasing the cost to block the system.

5 Implementation

We implemented CensorLess to validate our design and demonstrate its practicality. Our implementation consists of approximately 2,000 lines of code written in TypeScript, Node.js, and Python. While we primarily targeted AWS as our deployment platform due to its widespread adoption, the architecture is designed to be cloud-agnostic and can be deployed on alternative cloud providers with minimal modifications to the serverless function management components.

Our implementation incorporates several technical optimizations to ensure minimal latency and overhead. We implemented HTTP response streaming with request pipelining to reduce latency and improve throughput, particularly for web browsing applications. By distributing processing responsibilities between the local proxy and serverless bridge, we minimize the computational overhead at the serverless function layer, where execution time affects costs.

Serverless function bridge. The serverless function bridge represents the core of our circumvention system. We implemented this component with particular attention to minimizing resource consumption and execution time. The bridge was developed using Node.js with a codebase of less than 100 lines and containerized as a 146.7 MB Docker image to facilitate rapid deployment. We limited dependencies to only four external libraries, `axios`, `https`, `stream`, and `aws-sdk`, while implementing minimal request processing logic to maintain efficiency. The bridge also integrates a tagging mechanism for migration coordination by separately handling tag requests and general HTTP traffic within a single function.

To ensure security, each serverless function bridge operates according to the principle of least privilege. The security policy assigns only the essential AWS permissions required for operation: Amazon Resource Group Tagging API:TagResources and GetTagValues [2], Elastic Container Registry:GetRepositoryPolicy [3], and Lambda:InvokeFunctionUrl, CreateFunction, and ListFunctions [4]. This restricted permission set minimizes potential security vulnerabilities while maintaining full functionality.

The function refresher creates new serverless functions using the Docker image, deploys functions across diverse geographical regions, applies security policies to each created function, and removes expired bridges after completion of their usage cycle using the AWS Software Development Kit `boto3`. Access to serverless bridges and the use of `boto3` can only be performed by operators who own the serverless function pool.

Local proxy server. Clients interact with the system by running a local proxy server application implemented in TypeScript (under 400 lines of code). The local proxy server manipulates request headers from a user-defined port to convert HTTP requests to HTTPS, overcoming the restrictions of serverless platforms. It necessarily accepts the following header names: 'Cookie', 'User-Agent', 'Host', 'Content-Type', 'Permission-Policy', 'Accept', 'Accept-Encoding', and 'Accept-Language'. If the 'Strict-Transport-Security' header is included, the local proxy changes the value to 'max-age=0', and if 'Referer' or 'Origin' headers contain 'http:', it replaces it with 'https:'. After accepting the necessary headers, the local proxy redefines the `Host` as the serverless bridge and adds a new header called 'target-url', which indicates the true destination. Finally, the local proxy sends an HTTPS request to the serverless bridge with the reassembled headers. For adopting domain fronting, it sends requests to a subdomain with the true serverless bridge `Host` header. When the local proxy receives a response from the serverless bridge, it filters the acceptable request headers for consistency for the user and streams the data using pipelining. If the data includes 'https:', the local proxy replaces the string with 'http:'.

Protocol for private mode. As the private mode establishes a connection with the VPS server and uses a SOCKS proxy, it requires a protocol for an HTTPS request payload. The HTTPS payload is a message that communicates with the VPS server, and it includes the VPS server address in SOCKS address format, the server port, the payload length, and the encrypted payload. The encrypted payload consists of the client's public key, a nonce to prevent replay attacks, and the message. The VPS server interprets the message, which can be of several types. For a "start connection" message, the message contains the target address and the target port. A "data" message contains a signed connection ID, a compression flag to indicate whether the data is compressed or not, the original data length, the compressed length, and the data. A data message of length 0 is considered a "keepalive" message. A "close connection" message contains only a signature.

When it comes to the response message from the VPS server to the client encrypted with the client's public key, the response message also follows a defined message structure. For the "connection establishment" message, the response carries the type and

[2] https : / / docs.aws.amazon.com / service - authorization / latest / reference / list_amazonresourcegrouptaggingapi.html
[3] https : / / docs.aws.amazon.com / service - authorization / latest / reference / list_amazonelasticcontainerregistrypublic.html
[4] https : / / docs.aws.amazon.com / service - authorization / latest / reference / list_awslambda.html

connection ID. “Data response” message has the type, connection ID, compression flag, original data length, actual length, and data, and “connection close” message has the type, connection ID, message length, and message. If an error occurs, the message conveys the type, connection ID, error code, message length, and message.

In addition, the VPS filters private IP ranges to prevent Server-Side Request Forgery (SSRF) attacks. Connection attempts to the addresses in the loopback range, private networks, link-local addresses, multicast, broadcast, documentation, and unspecified ranges are rejected with an error code.

6 Evaluation

This section demonstrates that CensorLess achieves effective censorship circumvention while maintaining reasonable performance and low cost. We evaluate our serverless computing-based censorship circumvention system across four critical dimensions: system performance, serverless function performance as a bridge, operational costs, and censorship circumvention efficiency.

6.1 CensorLess Performance

To evaluate our systems' capability, we measured throughput with a single user under different content-loading scenarios with a Selenium script for consistency: reading articles from a web page with numerous objects, PDF downloads, and streaming video with actual browsing and interaction (clicking, downloading, and watching video). We compared the system's performance between without CensorLess, static serverless bridges in vanilla mode, bridges migrated every 20 seconds with a single client, and static serverless bridges in private mode. In practice, the migration cycle can be set longer than 20 seconds, consistent with the observed blocking period (in Section 6.4). Figure 3 reveals that CensorLess vanilla mode, regardless of migration, takes a longer time compared to without proxy and private mode in terms of the total execution time, even though the private mode requires more time for receiving the first response due to the secure connection setup process. The load-time waterfall graph for these use cases and each CensorLess mode is in Appendix A.1. Although requests from vanilla mode CensorLess route via the serverless bridge, it follows closely to the non-proxy throughput pattern, with negligible overhead and without significant performance penalty, while sacrificing privacy. Private mode creates sustained traffic even when the system is not actively loading content to maintain the SOCKS communication channel by polling. Comparing migration and non-migration scenarios, there are no considerable throughput pattern differences, with only slight increases in total execution time during migration periods. Since our system uses HTTPS to communicate with the serverless bridge, the migration does not require much time or processing steps to move the connectivity, and with and without migration show similar patterns across all three cases.

6.2 Serverless Function Bridge Performance

Our system's performance directly correlates with serverless function configuration. We evaluated three key configuration parameters: timeout, memory size, and ephemeral storage size. Setting a 15-second timeout proved optimal for all experiments, as shorter timeouts couldn't properly load streaming content while longer timeouts provided no considerable performance benefits. Similarly, ephemeral storage size affects temporary storage capacity rather than performance when memory allocation remains constant.

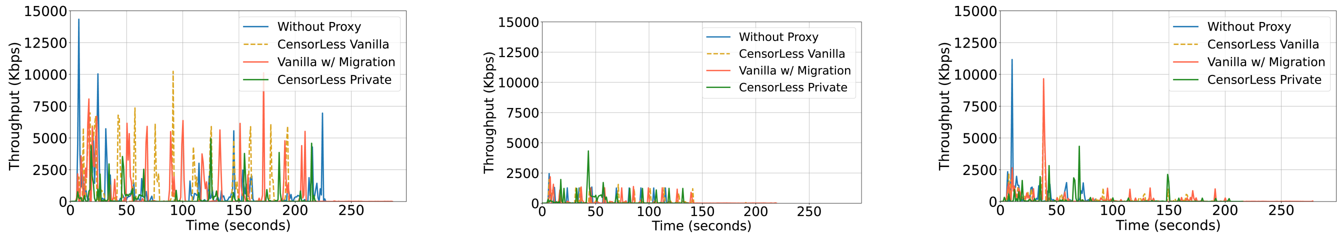
However, memory allocation impacts execution time. We measured throughput by repeating the same tasks (browsing a search engine, browsing a news website, and watching a short streamed video three times each) with different serverless function memory configurations. We conducted evaluations with varying memory allocations while keeping all other configurations constant (e.g., 15-second timeout and 512 MB default storage size). As shown in Figure 4, serverless functions with larger memory allocations (512 MB) demonstrated shorter total execution times compared to smaller allocations (128 MB, 256 MB) when performing identical tasks. This occurs because larger memory allocations allow the serverless function to store and process more data simultaneously. However, the serverless function bridge with a 15-second timeout, 512 MB default storage size, and 128 MB default memory size is sufficient in practical use, which does not cause conceivable degradation of performance and usability.

One of FaaS's greatest advantages is concurrency through autoscaling. Due to concurrency and autoscaling, serverless functions can support a stable service, and CensorLess is no exception. We evaluated our system's reliability under concentrated request loads on a single serverless bridge by analyzing average duration and success rates as concurrency indicators based on past experiment logs. As shown in Figure 5, during the initial phase (approximately 0-50 invocations), we observe notable volatility in both metrics. Average duration spikes to over 6000 ms in some functions, while the success rate occasionally drops below 90%. This initial instability stems from the cold start phenomenon common in serverless environments, where the first invocations to a newly migrated bridge require additional time for container initialization and deployment. The system stabilizes considerably between invocations 100-200, with average durations consistently below 1000 ms and success rates above 95%. This demonstrates the effectiveness of the FaaS platform's autoscaling capability, which dynamically allocates resources to handle increasing load. Although burst loads may occur infrequently, CensorLess's serverless bridge implemented on AWS Lambda by default can support stable service even at 1000 concurrent invocations [57]. This inherent scalability makes CensorLess reliable under heavy load.

6.3 Operational Costs

Cost efficiency is our system's highest priority. We compared monthly operational costs against existing economical circumvention tools, SpotProxy [54] and MassBrowser [64]. SpotProxy presents three scenarios of using spot instances: using the cheapest spot instances with single-NIC, with multi-NIC (they assumed 3 NICs, which is approximately 3 times cheaper than single-NIC), and using a static spot instance. Unlike MassBrowser, which requires persistent VMs for the operator, both SpotProxy and CensorLess stem primarily from their reliance on cloud services, SpotProxy on time-based pricing and CensorLess on request-based pricing.

For a consistent comparison, we followed SpotProxy's cost analysis assumption [54, §11.1], which is that each proxy processes 6.76 GB of traffic monthly. We then compare monetary costs based on



(a) Browsing website (cnn.com) with lots of objects.

(b) Downloading PDF files.

(c) Watching short streamed video.

Figure 3: This diagram shows throughput patterns according to different content loadings. The user browsed a news website and read 5 articles of different lengths (Figure 3a), browsed a research paper and downloaded the PDF file 10 times (Figure 3b), and the user watched the same video 5 times (Figure 3c).

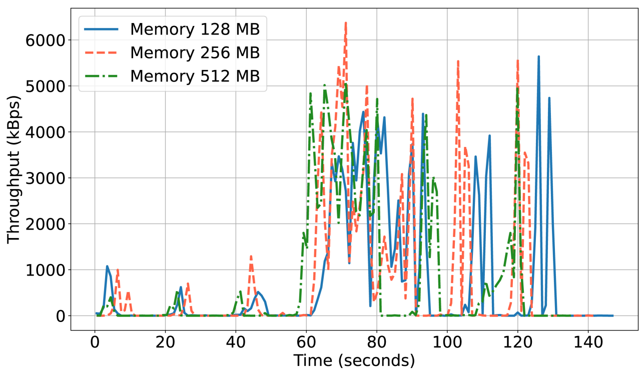


Figure 4: Results of throughput across different serverless function bridge configurations, with memory sizes ranging from 128 MB to 512 MB, a fixed timeout of 15 seconds, and ephemeral storage size of 512 MB.

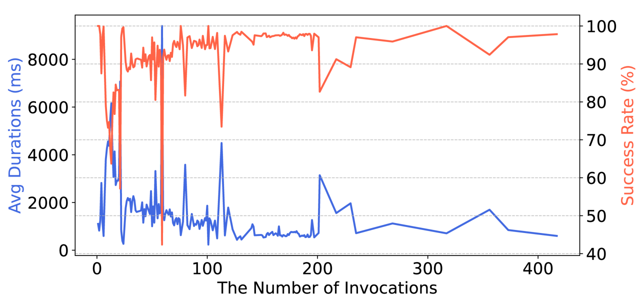


Figure 5: Results of the serverless function bridge's concurrency experiment. Based on our experimental log, we sorted the average duration and success rate according to the specific number of invocations.

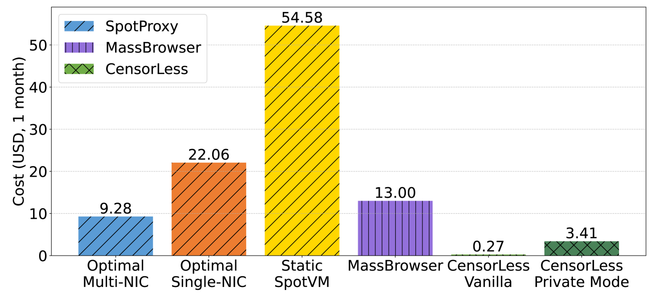


Figure 6: Cost comparison (1 month) results of a single proxy. Optimal multi-NIC, Optimal single-NIC, and Static SpotVM are from the SpotProxy paper[54]. As SpotProxy used actual operating costs, we calculated our cost based on our AWS bill and their assumptions.

each system's published cost analysis methodology and pricing model [9]. For our system, we calculated costs based on actual AWS Lambda billing, 0.00296 MB/request used. With 6.76 GB generating approximately 2,338,885 requests (6.76 GB / (0.00296 MB/request) = 2,338,885.14 requests) and AWS Lambda charging $0.20 per million requests, including 1 million free requests per month, our monthly cost is only $0.27 (with 1000 ms duration, 128 MB memory, and 512 MB storage). Since the private mode of our system additionally uses a VPS (EC2 in AWS), we used the `t4g.micro` (2 CPUs and 1 GB memory) EC2 instance. Based on the AWS pricing calculator, VPS costs $3.14 with EC2 Instance Savings Plans. In total, the monthly cost of the private mode is $3.41 ($0.27 + $3.14). As Figure 6 demonstrates, our approach achieves remarkable cost efficiency–34.4 times cheaper (97.1% savings) and even private mode, which is our most expensive mode, is 2.72 times cheaper (63.3% savings) than SpotProxy's most economical multi-NIC configuration.

We evaluated our system's cost scalability against SpotProxy. We assumed that each proxy handles one request per second sequentially for one hour, reflecting a scenario similar to SpotProxy's cost analysis [54, §11.1]. The spot instance used for SpotProxy is an `m6g.large`, the cheapest instance (at an hourly spot VM price of $0.0383) capable of running the SpotProxy program. SpotProxy is charged based on the hours the instances are in use, while our system is charged based on the number of requests. The number of proxy instances and requests increases at a constant rate. For the private mode, there is an additional charge for the single VPS (`t4g.micro` hourly costs $0.004). When the private mode serves with a single proxy, individual requests may be slower. The operator can mitigate this slower speed by adopting additional VPS instances,

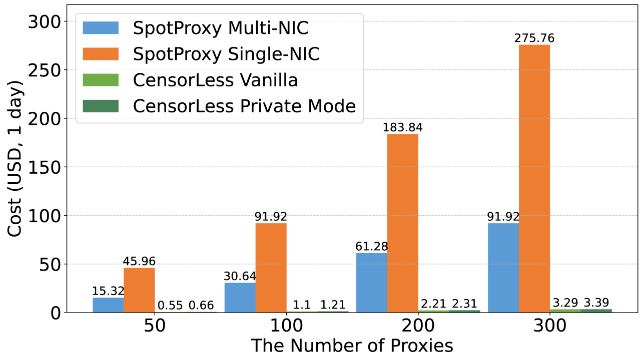


Figure 7: Cost comparison (1 day) between SpotProxy and CensorLess as the number of proxies increases. SpotProxy uses hour-based pricing, while CensorLess uses request-based pricing, so costs increase at a constant rate. (Private mode incurs an additional hour-based VPS charge.)

and the charged amount will increase. Figure 7 shows that our systems still cost less than \$3.50 even with 300 proxies. This suggests that leveraging FaaS is a more cost-effective option than using the cheapest IaaS service. Even with private mode, adopting more VPS instances to increase speed or enhance privacy still costs less than an IaaS service, with only a gradual and small linear increase (we demonstrate this in Appendix A.2).

6.4 Circumvention Effectiveness

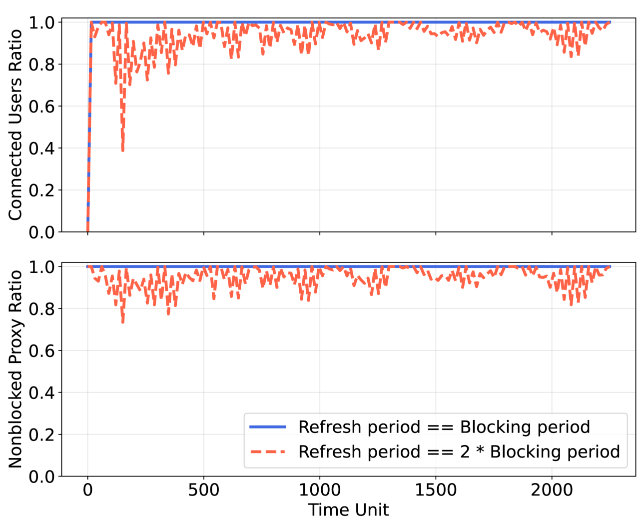


Figure 8: Censor simulation using the Optimal method. CensorLess preserves around 95% of connected users and non-blocked proxies, even when 50% of total clients are censor agents (when the refresh cycle is twice the censorship cycle).

6.4.1 ***Simulation***. We evaluated our URL refreshing method's censorship effectiveness using the state-of-the-art game-theoretic simulation framework for censorship circumvention proposed by Nasr et al. [63] This simulator employed three utility functions: client utility function ($U^t_{a_i}(p_x)$), proxy utility function ($U^t_{p_x}(a_i)$), and censor utility function (U^t_C) for client a_i, proxy p_x, and censor C. The client utility function (1) is a weighted sum of the proxy importance factors of the number of users who know the proxy ($B^t_{p_x}$), the number of users connected to the proxy ($c^t_{p_x}$), the total time utilization of the proxy $\tau^t_{p_x}$ and client locations (d_{a_i,p_x}). β_1, β_2, β_3, and β_4 are scaling factors that set the relative importance of proxy metrics, but we set all scaling factors to 1 in our evaluation.

$$U^t_{a_i}(p_x) = \beta_1 B^t_{p_x} + \beta_2 c^t_{p_x} + \beta_3 \tau^t_{p_x} - \beta_4 d_{a_i,p_x} \tag{1}$$

The proxy utility function (2) is a weighted sum of the client metrics, proxy utilization (T_{a_i}), the number of requests for new proxy addresses ($R^t_{a_i}$), the number of assigned blocked proxies ($\gamma^t_{a_i}$), the number of blocked proxies that a user knows ($\delta^t_{a_i}$), and client locations (d_{a_i,p_x}) with the objectives that assigning as many, unblocked, and long lived proxies. We set the scaling factors, α_1, α_2, α_3, α_4, and α_5, to 2, 1, 1, 2, and 10, respectively.

$$U^t_{p_x}(a_i) = (\alpha_1 min(T_{a_i}, \overline{T}) - \alpha_2 R^t_{a_i} - \alpha_3 \gamma^t_{a_i} - \alpha_4 \delta^t_{a_i} - \alpha_5 d_{a_i,p_x}) \tag{2}$$

The censor utility function (3) balances two competing objectives: proxy discovery and blocking impact.

$$U^t_C = \omega \sum_{a_i \in J} U^t_C(a_i) + r_{\text{Blocked}} \tag{3}$$

$\sum_{a_i \in J} U^t_C(a_i)$ represents the aggregate proxy discovery capability across all censoring agents J, r_{Blocked} denotes the fraction of censored clients unable to obtain working proxies, and ω is a weighting factor indicating the censor's strategic preference. The utility of censoring agent a_i, $U^t_C(a_i)$, is the average of a_i's scores by proxies (i.e., $\mathbb{E}_{p_x \in P}[U^t_{p_x}(a_i)]$). In the implementation, we followed SpotProxy's blocking capability that censors the proxy after 2 time units after discovery.

We applied these utility functions with minor implementation changes because our clients are assigned serverless bridge URLs, while this simulator distributes proxies to clients based on IP addresses. We assumed that each serverless bridge is paired with a single IP address, and the blocked URL cannot be used again, considering SNI-based blocking. Thus, our blocking mechanism is identical to IP-based blocking. To implement URL-based proxy assignment, we followed the serverless function's URL-generation rule; a string of fixed length randomly mixed with letters and numbers for a unique URL, particularly in AWS Lambda.

We evaluated our system's circumvention effectiveness by the ratio of connected users and non-blocked proxies against the optimal censor. The optimal censor uses an optimal game-theoretic approach as its censorship strategy, such as adjusting its waiting times for discovering new proxies and the frequency or intensity of its blocking efforts to maximize censoring effectiveness, aiming to uncover more proxies and block more clients. Setting 50% of clients as censor agents, we compared scenarios where refresh periods were either equal to or double the blocking periods. Every hyperparameter and configuration followed the SpotProxy's optimized values used for their evaluation [54, §8], and we compared the circumvention efficacy in Appendix A.3.1.

Against optimal game-theoretic censors (Figure 8), our system demonstrates even more stable and higher connectivity across both

Table 1: Website access results of CensorLess in the censored region of Nanjing, China.

Websites	Vanilla	Private Mode
google.com.et	✓[5]	✓
huffpost.com	✓	✓
thecitizen.in	✓	✓
chatgpt.org	✓	✓
gamestorrents.fm	✓	✓
voachinese.com	✓	✓
podcasts-online.org	✓	✓
biblechat.ai	✓	✓
chatdoc.com	✓	✓
vpn.com	✓	✓

client and proxy dimensions, even when using less frequent refreshing. To maximize reliability, we recommend setting migration cycles at least as frequently as expected censoring periods.

6.4.2 ***In censored region.*** We experimented with how CensorLess successfully circumvents censorship in the actual censored region of Nanjing, China. To test domain blocking, we collected 500 blocked domains in mainland China from the 5,000-domain DNS A experiment conducted by Sheffey et al. [76] The blocked domains tested included websites for search engines, AI tools, social media, news, entertainment, content sharing, pornography, etc. The current Chinese censorship pattern prevents users from receiving content from these websites over time. We sent 2 requests to each domain using a `curl` command, and measured whether the website returned an HTTP status code or no response within 100 seconds. We considered a website censored if there was no response within 100 seconds to explicitly distinguish between censored and response delay (usually, the status code was returned within 15 seconds). If the response was returned within 100 seconds, we sent the request to the next domain. Table 1 shows the results for 10 websites, with the 50 results provided in Appendix A.3, and the total results are available in our artifact. As shown in Table 1, both the CensorLess vanilla mode and the private mode receive responses to every request sent to blocked domains (the private mode fails to tubegays.xxx; we addressed the reason for failure in Appendix A.3, which is due to the website policy; reject request from AWS VPS).

This experiment was conducted in limited settings: a client of CensorLess was located in Autonomous System Number (ASN) AS45090 of Alibaba Cloud network in Nanjing, China, not in the residential networks for ethical reasons. During 36+ hours of in-region testing, our system, with 50 different serverless functions, was not blocked.

7 Security Discussion & Limitations

In this section, we discuss possible attack scenarios against CensorLess and corresponding defenses.

HSTS preloading. Serverless is at the core of designing CensorLess, and one inherent limitation of serverless is that providers prevent socket communication. Consequently, vanilla CensorLess relies on HTTP requests intercepted locally and translated to HTTPS. This approach encounters complications with modern websites that implement HTTP Strict Transport Security (HSTS) [24]. Web browsers maintain HSTS preloading lists that automatically redirect HTTP requests to HTTPS for specified domains before any network communication occurs. This client-side enforcement prevents CensorLess from capturing these requests through the local proxy, blocking access to websites on the HSTS preloading list.

There are two solutions to address the HSTS preloading issues. First, clients use CensorLess private mode, sacrificing the cost benefit of the vanilla system. Since the private mode local proxy uses a SOCKS proxy and establishes a TCP connection with the VPS, the client does not encounter the browser's force redirect from HTTP to HTTPS. Second, websites can be accessed through an earlier version of a web browser that does not have an HSTS preloading list. In early April 2025, when we conducted our experiments, every website was accessible in Firefox without HSTS preloading issues.

Cloud provider as an irrational actor. In Section 4.2, we assume cloud providers act rationally and will avoid measures that cause large collateral damage to their customers. We acknowledge, however, that a provider could instead collude with a censor and adopt countermeasures such as destination-based blocking or detection via traffic/operator-behavior pattern analysis. Monitoring the target destination (toward the public internet) could not effectively distinguish CensorLess's from other traffic because serverless functions are commonly used with third-party APIs on the internet or as web crawlers. Blocking via traffic analysis can be prevented with CensorLess deployed in private mode with multiple serverless bridges. While censors could limit CensorLess by detecting operator-behavior patterns (e.g., repeated function creation), this threat can be mitigated by varying the serverless bridge creation period and adopting the current bridge's URL update. Effectively, aggressive blocking aimed at CensorLess would also disrupt other cloud-based circumvention tools (e.g., Shadowsocks [21], V2Ray [83]) and many benign services, creating strong economic and reputational disincentives for providers. For these reasons, we model the provider as a rational actor. Moreover, CensorLess follows and is implemented based on the documented serverless features of Amazon Web Services (AWS).

Protocol support and HTTPS decapsulation. Due to serverless functions' restrictions on WebSocket communication, CensorLess supports only HTTPS GET and POST methods in vanilla mode, despite the widespread use of WebSocket connections in the modern web. Accordingly, CensorLess decapsulates HTTPS requests at the serverless bridge. Private mode mitigates these limitations at a slightly increased operational cost. In the private mode, we employed a custom proxying protocol that persistently tracks connection state across serverless bridge invocations, rather than standardized approaches such as HTTP CONNECT and MASQUE,

[5] It indicates the system successfully receives an HTTP status code from the website.

leveraging the fact that serverless functions accept traffic only via HTTP and stream responses within reasonable time/data bounds.

Traffic analysis attack. Censors can analyze HTTP patterns via modified headers or traffic patterns, but request headers sent to serverless functions are user-defined and variable. Identifying CensorLess traffic is difficult unless the cloud provider inspects payloads; CensorLess with private mode prevents such payload exposure through encryption. Since serverless functions are widely used as website domains or web crawlers, CensorLess traffic does not have unique patterns. IP-based blocking is also challenging for censors because serverless functions share public IP addresses. In future work, we will explore fingerprinting resistance techniques, such as User-Agent standardization [49].

Blocking via domain filtering. Censors could implement filtering based on characteristic domain patterns of serverless functions. For instance, AWS Lambda function URLs contain distinctive strings for the specific usage that could be used for blanket blocking. However, the widespread legitimate use of serverless functions for business applications creates substantial collateral damage from such broad filtering, making this approach less likely in practice. If such filtering emerges, techniques like DNS over HTTPS (DoH) [18] could provide an additional layer of obfuscation by encrypting DNS queries through an HTTPS encrypted session.

DoS attack. An attacker may send a large volume of connection requests to cause financial exhaustion, exploiting the request-based pay-as-you-go pricing model of the serverless function. This form of Denial-of-Service (DoS) attack is a known risk in serverless environments, where cloud providers throttle invocations as a countermeasure. At the service level, we mitigate DoS attacks by adding a client authentication step before user registration.

Cold start attack. Serverless functions experience cold starts [78, 84] when provisioning new execution environments, introducing latency and additional resource consumption. Malicious actors could exploit this by coordinating distributed bots to simultaneously trigger numerous cold starts, degrading performance and potentially inflating costs. As Ahmadi et al. note, mitigating such attacks requires limiting external access during cold start phases to enhance reliability [1]. Our design partially addresses this attack through minimal function code size and function refreshing, but remains vulnerable to sophisticated distributed attacks.

8 Future Work

Future work in this area could explore several promising directions. Performance enhancements could be developed to improve CensorLess's responsiveness under varying network conditions, including high-latency or congested networks common in censored regions. Further research could also explore adaptive bridge allocation strategies that dynamically respond to changes in censorship patterns and intensity. By collecting and analyzing real-time data on connection failures and successful circumventions, CensorLess could intelligently distribute traffic across regions and providers to maximize availability while minimizing costs.

Since CensorLess operates on a request-response model, it can integrate with other anti-censorship ecosystems. For example, it could support Tor bridge or Shadowsocks airport [20] distribution, serving as lightweight entry points into anonymity networks, or facilitate the secure distribution of the latest circumvention tool binaries such as Tor Browser [70], Psiphon [71], Lantern [59].

9 Conclusion

In this work, we present CensorLess, a novel serverless computing-based censorship circumvention system that inherits the benefits of serverless computing. Our approach addresses the fundamental challenges of existing circumvention proxies by prioritizing cost efficiency, censorship resistance, and performance.

CensorLess demonstrates that serverless functions offer significant advantages for censorship circumvention compared to traditional cloud-based approaches. Utilizing the ephemeral nature of serverless computing, our system can rapidly deploy and rotate bridges across different regions, making it difficult for censors to identify and block by IP address and domain name. The stateless design of serverless functions enables seamless migration between bridges without service interruption, providing continuous connectivity for users in censored regions. Additionally, we provide a privacy-preserving mode of CensorLess that supports tunneling over SOCKS with a cost-privacy trade-off. Notably, CensorLess can achieve 97% of cost savings through serverless computing's pay-as-you-go pricing based on the number of requests.

10 Ethical Considerations

For the real-world deployment evaluation of CensorLess in the censored region, we sent multiple requests to the known-blocked websites. In the whole experiment process, we adhere to comply with ethical standards. To minimize the risk, measurements were performed from an Alibaba Cloud server under our control rather than using personal devices or residential connections. Mitigating possible harm to the cloud provider, requests were limited to non-interactive curl probes checking response status only. Therefore, no participants were put at risk during our evaluation.

The vanilla CensorLess implementation reassembles HTTPS packets at serverless bridges, creating a trust relationship between users and bridge operators who have technical access to user traffic. We explicitly acknowledge this limitation and provide a privacy-preserving mode with encrypted channels at increased cost. As a consequence of CensorLess being available to the public, careful evaluation on operator's trustworthiness by users is required. We recommend operating serverless bridges for individual purposes or by trustworthy organizations with strict no-logging policies.

Acknowledgments

The work was supported in part by the NSF grant CNS-2333965, and by the Young Faculty Award program of the Defense Advanced Research Projects Agency (DARPA) under the grant DARPA-RA-21-03-09-YFA9-FP003. The views, opinions, and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

[1] Sina Ahmadi. 2024. Challenges and solutions in network security for serverless computing. *International Journal of Current Science Research and Review* 7, 01 (2024), 218–229.

[2] Alice, Bob, Carol, Jan Beznazwy, and Amir Houmansadr. 2020. How China Detects and Blocks Shadowsocks. In *Internet Measurement Conference*. ACM. https://censorbib.nymity.ch/pdf/Alice2020a.pdf
[3] Collin Anderson. 2013. *Dimming the Internet: Detecting Throttling as a Mechanism of Censorship in Iran*. Technical Report. University of Pennsylvania. https://arxiv.org/pdf/1306.4361v1.pdf
[4] Anonymous. 2014. Towards a Comprehensive Picture of the Great Firewall's DNS Censorship. In *Free and Open Communications on the Internet*. USENIX. https://www.usenix.org/system/files/conference/foci14/foci14-anonymous.pdf
[5] Anonymous. 2021. *How to Deploy a Censorship Resistant Shadowsocks-libev Server*. Retrieved January 2021 from https://gfw.report/blog/ss_tutorial/en/
[6] Anonymous and Amonymous. 2022. *Sharing a modified Shadowsocks as well as our thoughts on the cat-and-mouse game*. Retrieved February 2023 from https://github.com/net4people/bbs/issues/136
[7] Anonymous, Anonymous, Anonymous, David Fifield, and Amir Houmansadr. 2021. *A practical guide to defend against the GFW's latest active probing*. Retrieved April 2022 from https://github.com/net4people/bbs/issues/58
[8] Anonymous, Arian Akhavan Niaki, Nguyen Phong Hoang, Phillipa Gill, and Amir Houmansadr. 2020. Triplet Censors: Demystifying Great Firewall's DNS Censorship Behavior. In *Free and Open Communications on the Internet*. USENIX. https://www.usenix.org/system/files/foci20-paper-anonymous_0.pdf
[9] AWS. 2025. AWS Pricing Calculator — calculator.aws. https://calculator.aws/. [Accessed 17-04-2025].
[10] AWS. 2025. Configure Lambda function timeout - AWS Lambda — docs.aws.amazon.com. https://docs.aws.amazon.com/lambda/latest/dg/configuration-timeout.html [Accessed 05-06-2025].
[11] AWS. 2025. Generate a static outbound IP address using a Lambda function, Amazon VPC, and a serverless architecture - AWS Prescriptive Guidance — docs.aws.amazon.com. https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/generate-a-static-outbound-ip-address-using-a-lambda-function-amazon-vpc-and-a-serverless-architecture.html [Accessed 03-06-2025].
[12] AWS. 2025. What is AWS Lambda? - AWS Lambda — docs.aws.amazon.com. https://docs.aws.amazon.com/lambda/latest/dg/welcome.html [Accessed 03-06-2025].
[13] Kevin Bock, Yair Fax, Kyle Reese, Jasraj Singh, and Dave Levin. 2020. Detecting and Evading Censorship-in-Depth: A Case Study of Iran's Protocol Filter. In *Free and Open Communications on the Internet*. USENIX. https://www.usenix.org/system/files/foci20-paper-bock.pdf
[14] Kevin Bock, iyouport, Anonymous, Louis-Henri Merino, David Fifield, Amir Houmansadr, and Dave Levin. 2020. *Exposing and Circumventing China's Censorship of ESNI*. Retrieved September 2020 from https://github.com/net4people/bbs/issues/43#issuecomment-673322409
[15] Kevin Bock, Gabriel Naval, Kyle Reese, and Dave Levin. 2021. Even Censors Have a Backup: Examining China's Double HTTPS Censorship Middleboxes. In *Free and Open Communications on the Internet*. ACM. https://doi.org/10.1145/3473604.3474559
[16] Russell Brandom. 2018. Amazon Web Services starts blocking domain-fronting, following Google's lead — theverge.com. https://www.theverge.com/2018/4/30/17304782/amazon-domain-fronting-google-discontinued. [Accessed 22-04-2025].
[17] Chad Brubaker, Amir Houmansadr, and Vitaly Shmatikov. 2014. CloudTransport: Using Cloud Storage for Censorship-Resistant Networking. In *Privacy Enhancing Technologies Symposium*. Springer. https://petsymposium.org/2014/papers/paper_68.pdf
[18] Kimo Bumanglag and Houssain Kettani. 2020. On the impact of DNS over HTTPS paradigm on cyber systems. In *2020 3rd International Conference on Information and Computer Technologies (ICICT)*. IEEE, 494–499.
[19] Zimo Chai, Amirhossein Ghafari, and Amir Houmansadr. 2019. On the Importance of Encrypted-SNI (ESNI) to Censorship Circumvention. In *Free and Open Communications on the Internet*. USENIX. https://www.usenix.org/system/files/foci19-paper_chai_update.pdf
[20] Yi Ting Chua and Ben Collier. 2019. Fighting the "blackheart airports": internal policing in the Chinese censorship circumvention ecosystem. In *2019 APWG Symposium on Electronic Crime Research (eCrime)*. 1–9. https://doi.org/10.1109/eCrime47957.2019.9037500 ISSN: 2159-1245.
[21] Shadowsocks developers. 2025. Shadowsocks AEAD cihpher specification. https://shadowsocks.org/guide/aead.html
[22] VMess developers. 2025. VMess. https://www.v2fly.org/en_US/developer/protocols/vmess.html
[23] Roger Dingledine, Nick Mathewson, Paul F Syverson, et al. 2004. Tor: The second-generation onion router.. In *USENIX security symposium*, Vol. 4. 303–320.
[24] Ivan Dolnák and Ján Litvik. 2017. Introduction to HTTP security headers and implementation of HTTP strict transport security (HSTS) header for HTTPS enforcing. In *2017 15th International Conference on Emerging eLearning Technologies and Applications (ICETA)*. IEEE, 1–4.
[25] Frederick Douglas, Rorshach, Weiyang Pan, and Matthew Caesar. 2016. Salmon: Robust Proxy Distribution for Censorship Circumvention. *Privacy Enhancing Technologies* 2016, 4 (2016), 4–20. https://censorbib.nymity.ch/pdf/Douglas2016a.pdf
[26] Haixin Duan, Nicholas Weaver, Zongxu Zhao, Meng Hu, Jinjin Liang, Jian Jiang, Kang Li, and Vern Paxson. 2012. Hold-On: Protecting Against On-Path DNS Poisoning. In *Securing and Trusting Internet Names*. National Physical Laboratory. https://www.icir.org/vern/papers/hold-on.satin12.pdf
[27] Arun Dunna, Ciarán O'Brien, and Phillipa Gill. 2018. Analyzing China's Blocking of Unpublished Tor Bridges. In *Free and Open Communications on the Internet*. USENIX. https://www.usenix.org/system/files/conference/foci18/foci18-paper-dunna.pdf
[28] Agnieszka Dutkowska-Zuk, Austin Hounsel, Amy Morrill, Andre Xiong, Marshini Chetty, and Nick Feamster. 2022. How and why people use virtual private networks. In *31st USENIX Security Symposium (USENIX Security 22)*. 3451–3465.
[29] DW. 2024. DG8 leaders ask for more funding of censorship circumvention — corporate.dw.com. https://corporate.dw.com/en/dg8-leaders-call-for-more-funding-of-censorship-circumvention-tools/a-70846017 [Accessed 05-06-2025].
[30] Ed25519. 2025. Introduction — ed25519.cr.yp.to. https://ed25519.cr.yp.to/. [Accessed 20-11-2025].
[31] Kathrin Elmenhorst, Bertram Schütz, Nils Aschenbruck, and Simone Basso. 2021. Web censorship measurements of HTTP/3 over QUIC. In *Internet Measurement Conference*. ACM. https://dl.acm.org/doi/pdf/10.1145/3487552.3487836
[32] Roya Ensafi, David Fifield, Philipp Winter, Nick Feamster, Nicholas Weaver, and Vern Paxson. 2015. Examining How the Great Firewall Discovers Hidden Circumvention Servers. In *Internet Measurement Conference*. ACM. https://conferences2.sigcomm.org/imc/2015/papers/p445.pdf
[33] Shencha Fan, Jackson Sippe, Sakamoto San, Jade Sheffey, David Fifield, Amir Houmansadr, Elson Wedwards, and Eric Wustrow. 2025. Wallbleed: A Memory Disclosure Vulnerability in the Great Firewall of China. In *Network and Distributed System Security*. The Internet Society. https://gfw.report/publications/ndss25/data/paper/wallbleed.pdf
[34] Yuzhou Feng, Ruyu Zhai, Radu Sion, and Bogdan Carbunar. 2023. A Study of China's Censorship and Its Evasion Through the Lens of Online Gaming. In *USENIX Security Symposium*. USENIX. https://www.usenix.org/system/files/usenixsecurity23-feng.pdf
[35] Paul Ferguson and Geoff Huston. 1998. What is a VPN? (1998).
[36] Amos Fiat and Jared Saia. 2002. Censorship resistant peer-to-peer content addressable networks.. In *SODA*, Vol. 2. Citeseer, 94–103.
[37] David Fifield. 2018. Domain fronting to App Engine stopped working (#25804) · Issues · Legacy / Trac · GitLab — gitlab.torproject.org. https://gitlab.torproject.org/legacy/trac/-/issues/25804. [Accessed 22-04-2025].
[38] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. 2015. Blocking-resistant communication through domain fronting. *Privacy Enhancing Technologies* 2015, 2 (2015). https://www.icir.org/vern/papers/meek-PETS-2015.pdf
[39] David Fifield and Lynn Tsai. 2016. Censors' Delay in Blocking Circumvention Proxies. In *Free and Open Communications on the Internet*. USENIX. https://www.usenix.org/system/files/conference/foci16/foci16-paper-fifield.pdf
[40] Sergey Frolov, Jack Wampler, Sze Chuen Tan, J. Alex Halderman, Nikita Borisov, and Eric Wustrow. 2019. Conjure: Summoning Proxies from Unused Address Space. In *Computer and Communications Security*. ACM. https://jhalderm.com/pub/papers/conjure-ccs19.pdf
[41] Sergey Frolov and Eric Wustrow. 2019. The use of TLS in Censorship Circumvention. In *Network and Distributed System Security*. The Internet Society. https://tlsfingerprint.io/static/frolov2019.pdf
[42] Sergey Frolov and Eric Wustrow. 2020. HTTPT: A Probe-Resistant Proxy. In *Free and Open Communications on the Internet*. USENIX. https://www.usenix.org/system/files/foci20-paper-frolov.pdf
[43] ggailey777. 2025. Azure Functions scale and hosting — learn.microsoft.com. https://learn.microsoft.com/en-us/azure/azure-functions/functions-scale. [Accessed 19-04-2025].
[44] Google Cloud. 2025. Quotas | Cloud Run functions Documentation. https://cloud.google.com/functions/quotas. [Accessed 19-04-2025].
[45] Nguyen Phong Hoang, Jakub Dalek, Masashi Crete-Nishihata, Nicolas Christin, Vinod Yegneswaran, Michalis Polychronakis, and Nick Feamster. 2024. GFWeb: Measuring the Great Firewall's Web Censorship at Scale. In *USENIX Security Symposium*. USENIX. https://www.usenix.org/system/files/sec24fall-prepub-310-hoang.pdf
[46] Nguyen Phong Hoang, Arian Akhavan Niaki, Jakub Dalek, Jeffrey Knockel, Pellaeon Lin, Bill Marczak, Masashi Crete-Nishihata, Phillipa Gill, and Michalis Polychronakis. 2021. How Great is the Great Firewall? Measuring China's DNS Censorship. In *USENIX Security Symposium*. USENIX. https://www.usenix.org/system/files/sec21-hoang.pdf
[47] Amir Houmansadr, Giang T. K. Nguyen, Matthew Caesar, and Nikita Borisov. 2011. Cirripede: Circumvention Infrastructure using Router Redirection with Plausible Deniability. In *Computer and Communications Security*. ACM, 187–200. https://people.cs.umass.edu/~amir/papers/CCS11-Cirripede.pdf

[48] https://www.opentech.fund/news/author/ms-otf usr1/. 2025. Surge and Sustain Fund — opentech.fund. https://www.opentech.fund/funds/surge-and-sustain-fund/ [Accessed 05-06-2025].
[49] Tor Project Inc. 2025. Fingerprinting protections — tb-manual.torproject.org. https://tb-manual.torproject.org/anti-fingerprinting/. [Accessed 23-11-2025].
[50] Samrat Karak Jaiganesh Girinathan. 2022. Using Amazon CloudFront with AWS Lambda as origin to accelerate your web applications | Amazon Web Services. https://aws.amazon.com/blogs/networking-and-content-delivery/using-amazon-cloudfront-with-aws-lambda-as-origin-to-accelerate-your-web-applications/. [Accessed 06-06-2025].
[51] Jigsaw. 2025. *Outline.* https://getoutline.org/
[52] Divyank Katira, Gurshabad Grover, Kushagra Singh, and Varun Bansal. 2023. CensorWatch: On the Implementation of Online Censorship in India. In *Free and Open Communications on the Internet.* https://www.petsymposium.org/foci/2023/foci-2023-0006.pdf
[53] Patrick Tser Jern Kon, Aniket Gattani, Dhiraj Saharia, Tianyu Cao, Diogo Barradas, Ang Chen, Micah Sherr, and Benjamin E. Ujcich. 2024. NetShuffle: Circumventing Censorship with Shuffle Proxies at the Edge. In *Symposium on Security & Privacy.* IEEE. https://www.computer.org/csdl/pds/api/csdl/proceedings/download-article/1RjEabCelaM/pdf
[54] Patrick Tser Jern Kon, Sina Kamali, Jinyu Pei, Diogo Barradas, Ang Chen, Micah Sherr, and Moti Yung. 2024. SpotProxy: Rediscovering the Cloud for Censorship Circumvention. In *USENIX Security Symposium.* USENIX. https://www.cs-pk.com/sec24-spotproxy-final.pdf
[55] John Kristoff, Moritz Müller, Arturo Filastò, Max Resing, Chris Kanich, and Niels ten Oever. 2024. Internet Sanctions on Russian Media: Actions and Effects. In *Free and Open Communications on the Internet.* https://www.petsymposium.org/foci/2024/foci-2024-0001.pdf
[56] AWS Lambda. 2025. Lambda quotas. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html. [Accessed 19-04-2025].
[57] AWS Lambda. 2025. Understanding Lambda function scaling. https://docs.aws.amazon.com/lambda/latest/dg/lambda-concurrency.html. [Accessed 19-04-2025].
[58] Felix Lange, Niklas Niere, Jonathan von Niessen, Dennis Suermann, Nico Heitmann, and Juraj Somorovsky. 2025. I(ra)nconsistencies: Novel Insights into Iran's Censorship. In *Free and Open Communications on the Internet.* https://www.petsymposium.org/foci/2025/foci-2025-0002.pdf
[59] Lantern. 2025. Lantern — github.com. https://github.com/getlantern
[60] Colm MacCárthaigh. 2018. Enhanced Domain Protections for Amazon CloudFront Requests | Amazon Web Services — aws.amazon.com. https://aws.amazon.com/blogs/security/enhanced-domain-protections-for-amazon-cloudfront-requests/. [Accessed 22-04-2025].
[61] Nima Mahmoudi and Hamzeh Khazaei. 2022. Mlproxy: Sla-aware reverse proxy for machine learning inference serving on serverless computing platforms. *arXiv preprint arXiv:2202.11243* (2022).
[62] Garrett McGrath and Paul R Brenner. 2017. Serverless computing: Design, implementation, and performance. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW).* IEEE, 405–410.
[63] Milad Nasr, Sadegh Farhang, Amir Houmansadr, and Jens Grossklags. 2019. Enemy At the Gateways: Censorship-Resilient Proxy Distribution Using Game Theory.. In *NDSS.*
[64] Milad Nasr, Hadi Zolfaghari, Amir Houmansadr, and Amirhossein Ghafari. 2020. MassBrowser: Unblocking the Censored Web for the Masses, by the Masses. In *Network and Distributed System Security.* The Internet Society. https://www.ndss-symposium.org/wp-content/uploads/2020/02/24340.pdf
[65] Sadia Nourin, Van Tran, Xi Jiang, Kevin Bock, Nick Feamster, Nguyen Phong Hoang, and Dave Levin. 2023. Measuring and Evading Turkmenistan's Internet Censorship. In *The International World Wide Web Conference.* ACM. https://dl.acm.org/doi/abs/10.1145/3543507.3583189
[66] Aaron Ortwein, Kevin Bock, and Dave Levin. 2023. Towards a Comprehensive Understanding of Russian Transit Censorship. In *Free and Open Communications on the Internet.* https://www.petsymposium.org/foci/2023/foci-2023-0012.pdf
[67] P Pandiaraja and J Manikandan. 2015. Web proxy based detection and protection mechanisms against client based HTTP attacks. In *2015 international conference on circuits, power and computing technologies [ICCPCT-2015].* IEEE, 1–6.
[68] Diego Perino, Matteo Varvello, and Claudio Soriente. 2018. ProxyTorrent: Untangling the free HTTP (S) proxy ecosystem. In *Proceedings of the 2018 World Wide Web Conference.* 197–206.
[69] Tor Project. 2018. [tor-project] Summary of meek's costs, March 2018 — archive.torproject.org. https://archive.torproject.org/websites/lists.torproject.org/pipermail/tor-project/2018-April/001743.html [Accessed 05-06-2025].
[70] Tor Project. 2025. The Tor Project | Privacy & Freedom Online — torproject.org. https://www.torproject.org/ [Accessed 05-06-2025].
[71] Psiphon Inc. 2025. Psiphon-Automation: Circumvention System Automation and Server Management. https://github.com/Psiphon-Inc/psiphon-automation Accessed: 2025-02-01.
[72] Raymond Rambert, Zachary Weinberg, Diogo Barradas, and Nicolas Christin. 2021. Chinese Wall or Swiss Cheese? Keyword filtering in the Great Firewall of China. In *WWW.* ACM. https://censorbib.nymity.ch/pdf/Rambert2021a.pdf
[73] Reethika Ramesh, Ram Sundara Raman, Apurva Virkud, Alexandra Dirksen, Armin Huremagic, David Fifield, Dirk Rodenburg, Rod Hynes, Doug Madory, and Roya Ensafi. 2023. Network Responses to Russia's Invasion of Ukraine in 2022: A Cautionary Tale for Internet Freedom. In *USENIX Security Symposium.* USENIX. https://censoredplanet.org/assets/russia-ukraine-invasion.pdf
[74] Reethika Ramesh, Anjali Vyas, and Roya Ensafi. 2023. " All of them claim to be the best": Multi-perspective study of {VPN} users and {VPN} providers. In *32nd USENIX Security Symposium (USENIX Security 23).* 5773–5789.
[75] GFW Report. 2022. *Large scale blocking of TLS-based censorship circumvention tools in China.* Retrieved October 2022 from https://github.com/net4people/bbs/issues/129
[76] Jade Sheffey, Ali Zohaib, Dayeon Kang, Zakir Durumeric, Amir Houmansadr, and Qiang Wu. 2025. I'll Shake Your Hand: What Happens After DNS Poisoning. *Free and Open Communications on the Internet* (2025).
[77] Signal. 2018. A letter from Amazon — signal.org. https://signal.org/blog/looking-back-on-the-front/. [Accessed 22-04-2025].
[78] Paulo Silva, Daniel Fireman, and Thiago Emmanuel Pereira. 2020. Prebaking functions to warm the serverless cold start. In *Proceedings of the 21st International Middleware Conference.* 1–13.
[79] Microsoft Security Team. 2021. Securing our approach to domain fronting within Azure. https://www.microsoft.com/en-us/security/blog/2021/03/26/securing-our-approach-to-domain-fronting-within-azure/. [Accessed 23-04-2025].
[80] The4. 2025. What Are the 9 Operating Costs of a Virtual Private Network Provider? — businessplan-templates.com. https://businessplan-templates.com/blogs/running-costs/virtual-private-network-provider [Accessed 05-06-2025].
[81] Finbarr Toesland. 2022. Lantern, US-backed censorship circumvention tool records growth — techerati.com. https://www.techerati.com/news-hub/us-backed-censorship-circum-tool-lantern-records-growth/. [Accessed 05-06-2025].
[82] trojan developers. 2025. *trojan.* https://github.com/trojan-gfw/trojan
[83] V2Ray developers. 2025. *V2Ray.* https://github.com/v2fly/v2ray-core
[84] Parichehr Vahidinia, Bahar Farahani, and Fereidoon Shams Aliee. 2020. Cold start in serverless computing: Current trends and mitigation strategies. In *2020 International Conference on Omni-layer Intelligent Systems (COINS).* IEEE, 1–7.
[85] Erwin Van Eyk, Lucian Toader, Sacheendra Talluri, Laurens Versluis, Alexandru Uță, and Alexandru Iosup. 2018. Serverless is more: From paas to present cloud computing. *IEEE Internet Computing* 22, 5 (2018), 8–17.
[86] Nicholas Weaver, Christian Kreibich, Martin Dam, and Vern Paxson. 2014. Here be web proxies. In *International Conference on Passive and Active Network Measurement.* Springer, 183–192.
[87] Tim Wilde. 2012. *Knock Knock Knockin' on Bridges' Doors.* Retrieved February 2023 from https://blog.torproject.org/blog/knock-knock-knockin-bridges-doors
[88] Philipp Winter. 2013. *GFW actively probes obfs2bridges.* Retrieved February 2020 from https://bugs.torproject.org/8591
[89] Philipp Winter and Stefan Lindskog. 2012. How the Great Firewall of China is Blocking Tor. In *Free and Open Communications on the Internet.* USENIX. https://www.usenix.org/system/files/conference/foci12/foci12-final2.pdf
[90] Mingshi Wu, Jackson Sippe, Danesh Sivakumar, Jack Burg, Peter Anderson, Xiaokang Wang, Kevin Bock, Amir Houmansadr, Dave Levin, and Eric Wustrow. 2023. How the Great Firewall of China Detects and Blocks Fully Encrypted Traffic. In *USENIX Security Symposium.* USENIX. https://www.usenix.org/system/files/sec23fall-prepub-234-wu-mingshi.pdf
[91] Diwen Xue, Anna Ablove, Reethika Ramesh, Grace Kwak Danciu, and Roya Ensafi. 2024. Bridging Barriers: A Survey of Challenges and Priorities in the Censorship Circumvention Landscape. In *USENIX Security Symposium.* USENIX. https://www.usenix.org/system/files/usenixsecurity24-xue-bridging.pdf
[92] Diwen Xue, Benjamin Mixon-Baca, ValdikSS, Anna Ablove, Beau Kujath, Jedidiah R. Crandall, and Roya Ensafi. 2022. TSPU: Russia's Decentralized Censorship System. In *Internet Measurement Conference.* ACM. https://dl.acm.org/doi/pdf/10.1145/3517745.3561461
[93] Diwen Xue, Reethika Ramesh, ValdikSS, Leonid Evdokimov, Andrey Viktorov, Arham Jain, Eric Wustrow, Simone Basso, and Roya Ensafi. 2021. Throttling Twitter: an emerging censorship technique in Russia. In *Internet Measurement Conference.* ACM. https://dl.acm.org/doi/pdf/10.1145/3487552.3487858
[94] Tony Huiquan Zhang, Jianhua Xu, and Jinjin Liu. 2024. How Do Toothless Tigers Bite? Extra-institutional Governance and Internet Censorship by Local Governments in China. *The China Quarterly* 2024 (2024). Issue First View. https://www.cambridge.org/core/services/aop-cambridge-core/content/view/B1BB347F7458EBF033A65461D1C2D82A/S0305741024000602a.pdf
[95] Zhensheng Zhang, Ya-Qin Zhang, Xiaowen Chu, and Bo Li. 2004. An overview of virtual private network (VPN): IP VPN and optical VPN. *Photonic network communications* 7 (2004), 213–225.
[96] Hadi Zolfaghari and Amir Houmansadr. 2016. Practical Censorship Evasion Leveraging Content Delivery Networks. In *Computer and Communications Security.* ACM. https://people.cs.umass.edu/~amir/papers/CDNReaper.pdf

A Additional Experiments

A.1 CensorLess Performance

In Figure 9, each subfigure shows the time to load the content in three different cases: browsing the cnn.com webpage, downloading a PDF file, and watching a short video. The Y label exhibits the destination where the request goes, and ↻ indicates the reuse connection to the same destination. While the waterfall result without a proxy can specify the destination, the result with CensorLess successfully hides the destination of requests. Vanilla mode CensorLess without and with bridge migration have similar load time patterns because they use HTTPS GET/POST methods only. In Figure 9h, vanilla with migration seamlessly handles the client requests even though the bridge migration happened between 0.5 and 1.0 second. Notably, CensorLess private mode requires more time to load the webpage because it routes traffic to an additional VPS and goes through a connection setup process for secure communication.

A.2 Operational Costs

We analyzed how the operational cost of a single proxy changes depending on the security level. Since the private mode incurs an additional cost for the VPS based on hourly usage, we measured the security level as the number of hours of using the private mode. 0% of security level indicates the client uses only vanilla CensorLess for 24 hours, 25% indicates the client uses the private mode for 6 hours out of 24 hours, and 100% indicates the entire traffic for 24 hours is in private mode. While the state-of-the-art cost-efficient tool, SpotProxy, uses spot instances (cost-saving IaaS instances) for individual proxies, vanilla CensorLess uses request-based serverless functions for individual proxies and additionally adopts the IaaS instance (i.e., VPS) for the private mode. In our experiment, this VPS requires a lower configuration (2 CPUs and 1 GB memory) compared to SpotProxy (2 CPUs and 8 GB memory), resulting in a lower hourly cost. Figure 10 demonstrates a gradual and small linear increase in costs as the usage of the private mode increases. Even at 100% (i.e., private mode for 24 hours), the security level charges less than SpotProxy's cheapest scenario. The cost of the private mode primarily depends on the VPS cost, similar to SpotProxy; however, a single VPS in private mode supports multiple serverless proxies, saving costs as the number of proxies increases.

When analyzing serverless function economics at scale, we discovered that cost linearity breaks at higher concurrency levels due to varying function durations. Table 2 shows that lower invocation numbers actually consume more time due to cold starts, affecting overall costs since duration impacts billing.

We investigated the serverless bridge operational cost as the number of proxies and clients increases. In Figure 11, each bar in the same position within each group represents the same number of invocations when a client sends to one proxy. Due to the relatively long duration at the lower invocations, the costs of 10 invocations/client and 50 invocations/client are similar. The serverless function charges a price increase of around 3.6 times when a client sends 100 requests to one proxy. This means maintaining moderate invocation levels (around 50 per client) with more proxies is more cost-efficient than fewer proxies handling burst traffic, when the total number of function calls remains equal. This insight provides valuable guidance for optimal system deployment in practice.

Table 2: Average Serverless Function Duration by Different Number of Concurrent Invocations

Invocations	Duration (ms)
10	4557.69
50	1002.35
100	1863.76
200	720.85

A.3 Circumvention Effectiveness

A.3.1 ***Simulation.***

We evaluated our system's circumvention effectiveness on both the client and proxy sides against two types of censors: aggressive and optimal. Aggressive censor agents immediately block any new proxy that they identify, while the optimal censor uses an optimal game-theoretic approach as its censorship strategy. Setting 50% of clients as censor agents, we compared scenarios where refresh periods were either equal to or double the blocking periods. As shown in Figure 12, our system shows large fluctuations in the initial phases for both connected users and nonblocked proxy ratios when the refresh period was set to double the blocking period, while matching the refresh period with the blocking period shows stable connections. Even with 50% of users being aggressive censor agents, CensorLess guarantees an average of 70% connectivity to users.

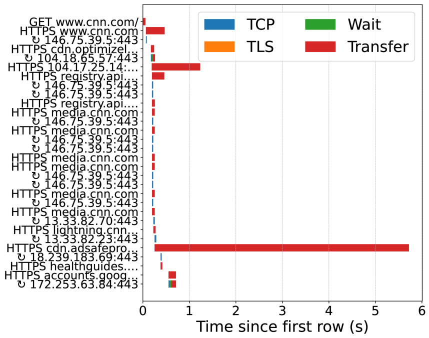
CensorLess demonstrates censorship circumvention efficacy comparable to the state-of-the-art IP-based tool (e.g., SpotProxy), as it resists censorship by rotating proxies using URLs. In this simulation, the censor blocks users and proxies with identified URLs, following the same approach as the IP-based proxy assignment algorithm. As shown in Figure 13, although the dominating algorithm varies over time, both provide identical ratios of connected users on average—77% with the aggressive method and 99% with the optimal method.

A.3.2 ***In censored region.*** We experimented with 500 blocked websites in mainland China, a well-known censored region. The part of the results, against 50 blocked domains, is shown in Table 3. The vanilla CensorLess successfully receives a response for every request sent to those domains, while the private mode only fails to receive responses from tubegays.xxx. The failure of tubegays.xxx in private mode is due to AWS VPS's request being denied by the website policy. While the final request to the website is from AWS VPS, the vanilla mode CensorLess successfully received the response since it uses serverless function.

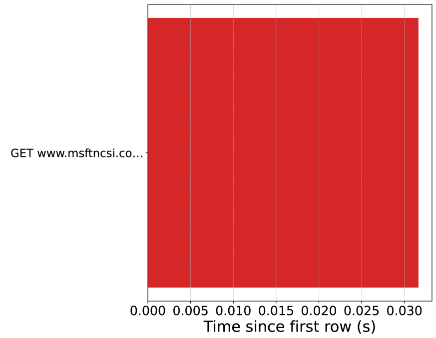
[6] It indicates the system did not receive any response from the website.

Table 3: Complete website access results of CensorLess in the censored region, Nanjing, China

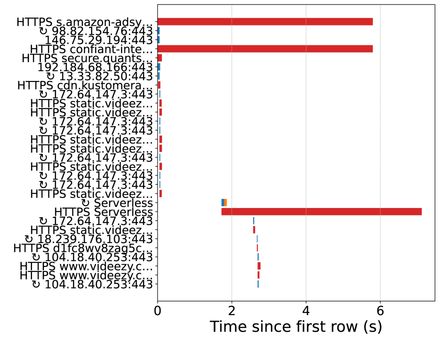
Websites	Vanilla	Private	Websites	Vanilla	Private
jpc.de	✓	✓	huffpost.com	✓	✓
ddd-smart.net	✓	✓	pornstars.tube	✓	✓
google.com.et	✓	✓	fosstodon.org	✓	✓
njav.tv	✓	✓	lohaco.jp	✓	✓
popyard.space	✓	✓	asiafinancial.com	✓	✓
google.co.hu	✓	✓	red-movies.com	✓	✓
chatdoc.com	✓	✓	nifty.org	✓	✓
dergipark.org.tr	✓	✓	domai.com	✓	✓
ero-labs.com	✓	✓	factmandu.com	✓	✓
wikipedia.org	✓	✓	rfa.org	✓	✓
nationalfile.com	✓	✓	getlink.pro	✓	✓
gofundme.com	✓	✓	erotic-hentai.com	✓	✓
indiandefencereview.com	✓	✓	thecitizen.in	✓	✓
thumbnailseries.com	✓	✓	simplex.im	✓	✓
voacantonese.com	✓	✓	tubegays.xxx	✓	×[6]
hostux.net	✓	✓	beliefnet.com	✓	✓
flyingjizz.com	✓	✓	mastodon.world	✓	✓
gamestorrents.fm	✓	✓	xfrenchies.com	✓	✓
epochtimes.com.ua	✓	✓	aflegal.org	✓	✓
ejecentral.com.mx	✓	✓	aiweiwei.com	✓	✓
flyflv.com	✓	✓	voyeurweb.com	✓	✓
voachinese.com	✓	✓	memeorandum.com	✓	✓
swag.live	✓	✓	chat-gpt.org	✓	✓
modelhub.com	✓	✓	newsdirectory3.com	✓	✓
ideapocket.com	✓	✓	hsav.xyz	✓	✓



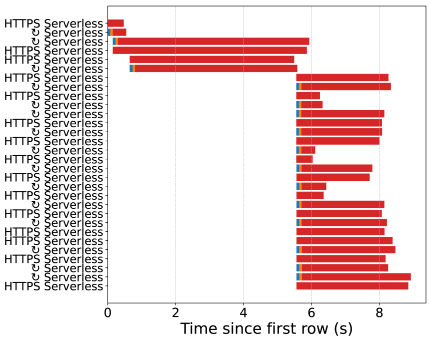
(a) cnn.com browsing: without proxy



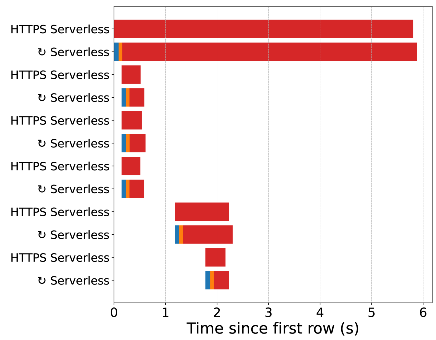
(b) PDF downloading: without proxy



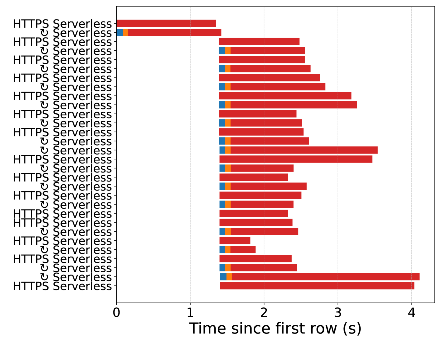
(c) Watching Video: without proxy



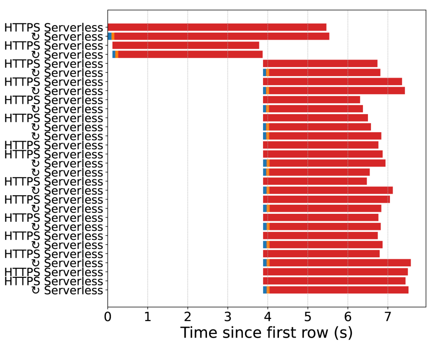
(d) cnn.com browsing: vanilla



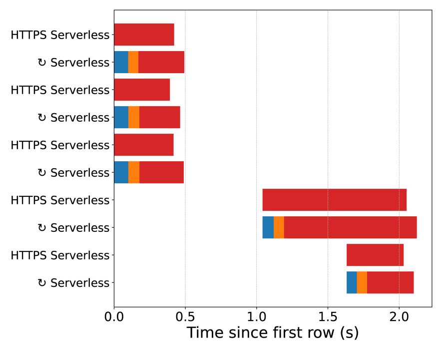
(e) PDF downloading: vanilla



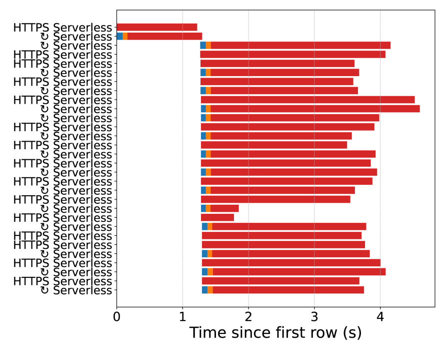
(f) Watching Video: vanilla



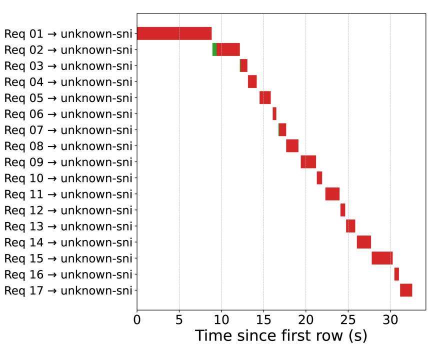
(g) cnn.com browsing: vanilla with migration



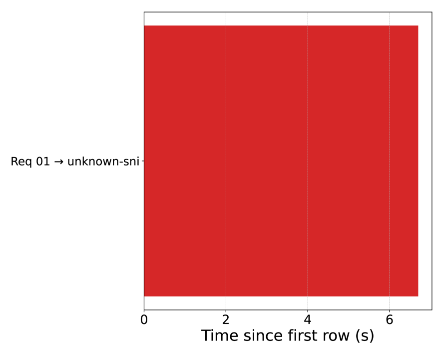
(h) PDF downloading: vanilla with migration



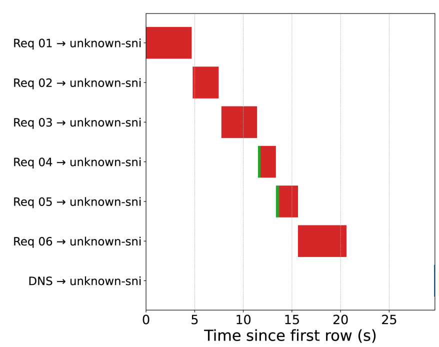
(i) Watching Video: vanilla with migration



(j) cnn.com browsing: private mode



(k) PDF downloading: private mode



(l) Watching Video: private mode

Figure 9: Baseline comparison for three use cases: loading the cnn.com page, which has many objects, downloading a PDF file, and watching a short video.

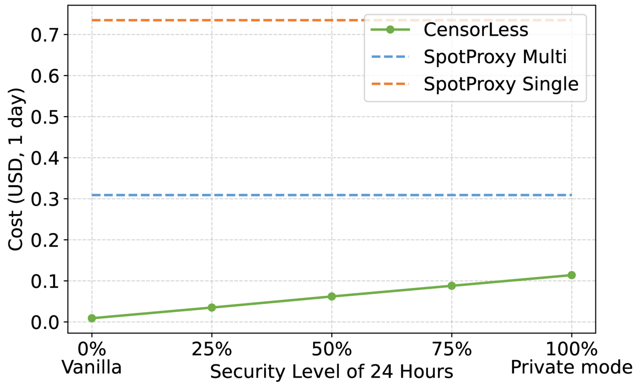


Figure 10: Cost comparison result by the security level of a single proxy. 25% means that 25% of the total traffic over the 24 hours (6 hours) is used in private mode.

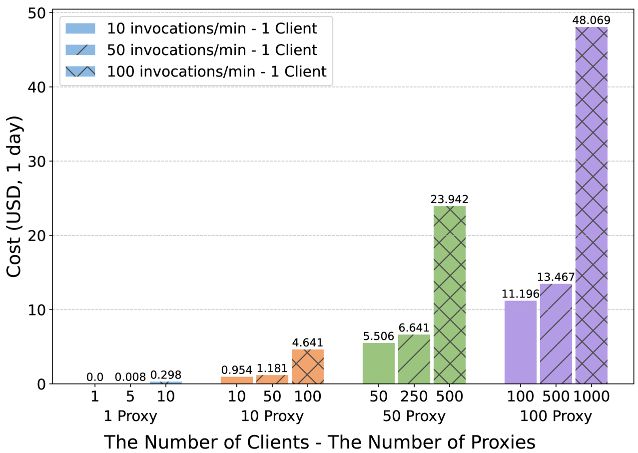


Figure 11: Serverless bridge cost scaling (1 day) results when the number of clients and proxies differ. Bars in each group indicate the different numbers of clients. Costs are compared when the number of invocations sent per minute by one client within each group varies to 10, 50, and 100.

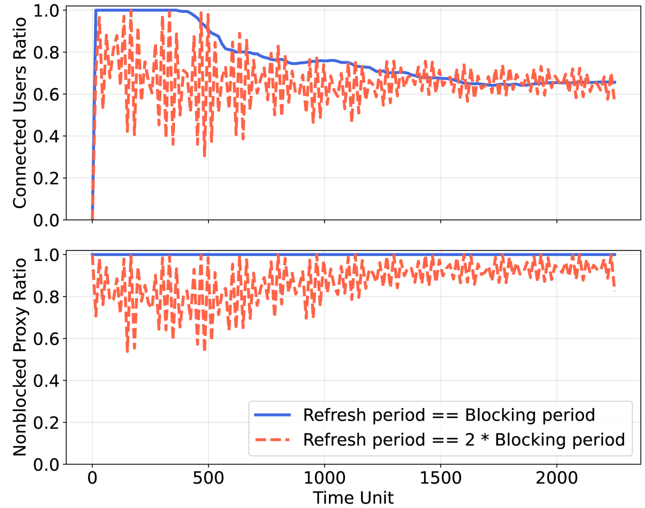


Figure 12: Censor simulation using the Aggressive method. CensorLess preserves 66% of connected users and 87% of non-blocked proxies, even when 50% of total clients are censor agents (when the refresh cycle is twice the censorship cycle).

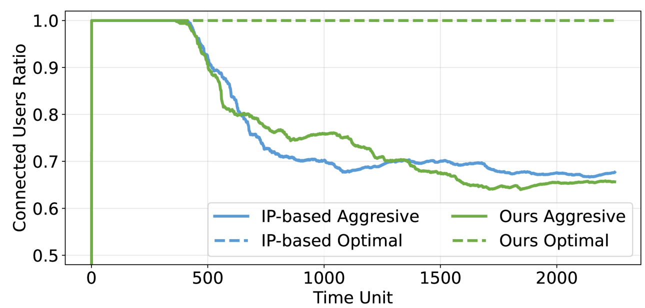


Figure 13: Censor simulation comparison. URL-based (i.e., CensorLess) and IP-based proxy rotating tool present similar censor resistance results (77% connected users with the Aggresive method) when 50% of total clients are censor agents and the refresh cycle is the same as censor cycle.