Free and Open
COMMUNICATIONS
https://foci.community on the Internet

The Use of Push Notification in Censorship Circumvention

Diwen Xue
University of Michigan

Roya Ensafi
University of Michigan

Free and Open Communications on the Internet 2023(1), 22-32

The Use of Push Notification in Censorship Circumvention

Diwen Xue Roya Ensafi

University of Michigan

Abstract

Push notifications provide a way for applications to deliver time-sensitive information directly to users. In recent years, they have gained widespread adoption across mobile and desktop platforms. In this paper, we explore the use of push notification services for censorship circumvention. Supported with measurements, we argue that push notifications offer high availability, as blocking them would incur significant collateral damage, making them ideal candidates to tunnel circumvention traffic.

We present two censorship circumvention systems that leverage push notification as a transport. *PushRSS* is a blocking-resistant content aggregator that tunnels RSS updates through a push notification network. Once bootstrapped, the tool remains operational even if the server IP is outright blocked. *PushProxy* is a general-purpose proxy that routes user's downstream traffic through a push notification service, while keeping upstream an independent channel. By decoupling the downstream from upstream, *PushProxy* mitigates the ability of network adversaries to perform per-flow traffic analysis, while providing performance comparable to popular symmetric proxies. Although these systems have their limitations, we believe push notification still holds potential as a circumvention transport that complements existing approaches.

1 Introduction

ISPs, advertisers, and national governments are increasingly disrupting, manipulating, and monitoring Internet traffic. With authoritarianism at rise, more networks have deployed censorship policies, as the free flow of information and exchange of ideas on the Internet have been perceived as a threat by repressive regimes. In response, numerous circumvention tools have been proposed and implemented over the years [2, 6, 10, 13, 26, 27, 32, 33, 45]. As censorship measures continue to advance alongside technology and detection methods, it is crucial for tool designers to continually explore and experiment with novel methods for circumventing censorship.

In this work, we explore the use of push notification as a transport for circumvention traffic. Push notification services, also referred to as cloud messaging, enable applications to transmit time-sensitive information directly to users. These

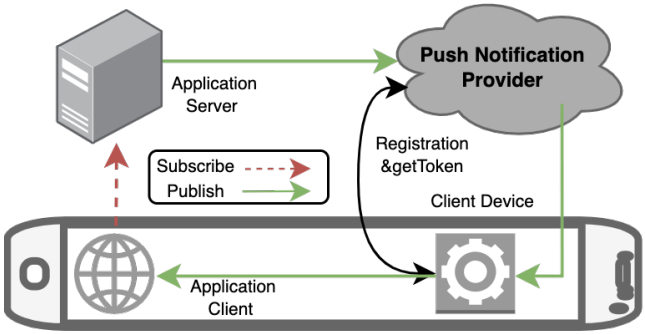

Figure 1: **Workflow of Push Notification Services.**

services have seen widespread adoptions in recent years, with over 600 billion messages sent to 2 billion users in H1 2021 from one provider alone [4]. Examples of providers include Google Firebase Cloud Messaging (GCM) and Apple Push Notification Service (APNS). These providers offer a cost-effective and highly customizable solution for delivering user-specific contents directly to the targeted app.

Push notification has become a widely integrated feature in both mobile and desktop applications, and there is currently no alternative mechanism that offers a similar functionality. As a result, blocking push notification services outright can cause significant collateral damage for state-level censors, which is a common criterion for effective censorship circumvention [40]. Compared to other circumvention systems, however, push notification services typically conform to a Publish/Subscribe model (Figure 1) and have limitations on the traffic patterns they support. Despite this, certain types of applications could still improve their availability by tunneling traffic through push notification providers.

In essence, push notification services function as one-way channels that relay information from an app server to an app client. Importantly, they do not require any direct connection between the server and the client, and is expected to remain operational even if the app itself is censored down to IP level. We present two circumvention systems that leverage the blocking-resistant property of such channels. First, *PushRSS* is a content aggregator on Android that enables users to subscribe to structured feeds (e.g. RSS) and receive updates tunneled through push notification networks. We provide details on its design and explore potential applications,

both as a standalone system and as an add-on that streamlines the bootstrapping process of existing circumvention tools. Additionally, we design and implement *PushProxy*, a general-purpose asymmetric proxy written in Go that routes downstream traffic through push notification services, while keeping upstream an independent channel. Such decoupling provides additional defense against traffic analysis compared to symmetric proxies, as it reduces the information an attacker with a limited network perspective can learn or correlate.

We evaluate the potential of push notification as a transport for circumvention traffic in terms of both availability and performance. We conduct large-scale, longitudinal measurements to test for potential censorship practices against push notification services across regions and networks. Our findings indicate that only a small number of ASes actively interfere with push notification connections over HTTPS, with the majority of blocking incidents taking place in China during politically sensitive times. Interestingly, we also discovered evidence indicating that the Chinese censors were wary of outright blocking push notification services in the long run and had made whitelist exceptions after only days of blocking, presumably due to the substantial collateral damage that would result. We evaluate the performance of push notification as a downstream transport by comparing *PushProxy* with symmetric tools (OpenVPN/Shadowsocks) in a realistic setting. Our analysis demonstrated a comparable performance between these tools in terms of both latency and bandwidth. While there are certain limitations, we believe that push notification can still serve as a viable circumvention transport that complements existing approaches.

2 Push Notification for Circumvention

Push notification services enable applications to deliver time-sensitive messages to user devices, relayed by service providers' networks. Examples of such services include Google Firebase Cloud Messaging (GCM/FCM), Apple Push Notification Service (APNS), Microsoft Push Notification Service, *etc.*. Fundamental to many mobile and desktop apps that require timely alert to users of updates or new messages, push notification services have seen significant growth in implementation and usage, with a recent article showing that over 600 billion messages were sent to over 2 billion users in six months from one provider alone [4].

Users may have the misconception that push notifications are sent directly by application servers. However, these messages are in fact sent from application servers to push notification providers, who subsequently deliver them to the targeted user devices. The exact implementation differs from provider to provider, but they generally conform to a Publish/Subscribe model, as shown in Figure 1. First, a newly installed app needs to undergo an enrollment procedure by reaching out to a push notification provider. Following this, the client device acquires a device- and app-specific registration token. Next, the app client registers with the app server, providing its registration token and subscription information such as username, groupname, or interested topics. In order to send a notification, the application server sends a request to the push notification provider, which is indexed by the registration token belonging to the targeted client, and also includes notification payloads.

While several previous works have explored the security and privacy aspects of the push notification ecosystem [11, 29, 30, 49], to the best of our knowledge, we are the first to explore the use of push notification in the context of Internet censorship. Our study is motivated by the following considerations, which have previously guided the development of other circumvention systems [40]:

High Collateral Damage Push notifications have been widely integrated into mobile and desktop apps as a means to reach to their users. There's no other mechanism that offers a similar functionality. Blocking a push notification service would result in noticeable effects from end-users and would affect all apps relying on the service.

Low Latency, Low cost, Mid Bandwidth Push notification services operate in real time to deliver time-sensitive information that requires user's attention. Both FCM and APNS allow a single push message to carry up to 4 KiB of payload and can be directly processed by the targeted app, without visible prompts to users. The service is offered at no additional cost[1], a significant advantage over other high-availability systems using cloud providers or blockchain [10, 20, 22].

Plausible Fingerprints Compared to circumvention methods that rely on mimicry or randomization for obfuscation, which have been shown to be flawed in practice [25, 47], push notification-based systems offer more realistic fingerprints as the traffic is tunneled within legitimate connections to the actual service endpoints. Moreover, as push messages are TLS-encrypted, DPIs cannot block by keyword or destination app.

On the other hand, the Publish/Subscribe model of push notification has certain limitations, as it restricts the types of communication patterns that can be supported. Specifically, it is most suitable for scenarios where the volume and frequency of downstream traffic are asymmetrically higher than that of upstream traffic, such as content aggregation services or web browsing, where outbound requests are often lighter than inbound responses. In the next sections, we present two tools that leverage push notification as a circumvention transport.

3 *PushRSS*: Blocking-resistant Aggregator

We first present *PushRSS*, a blocking-resistant RSS aggregator that tunnels content updates through push notification services. An RSS Aggregator is a content distribution service that allows users to subscribe to various RSS feeds, collects

[1] A developer subscription is needed for APNS

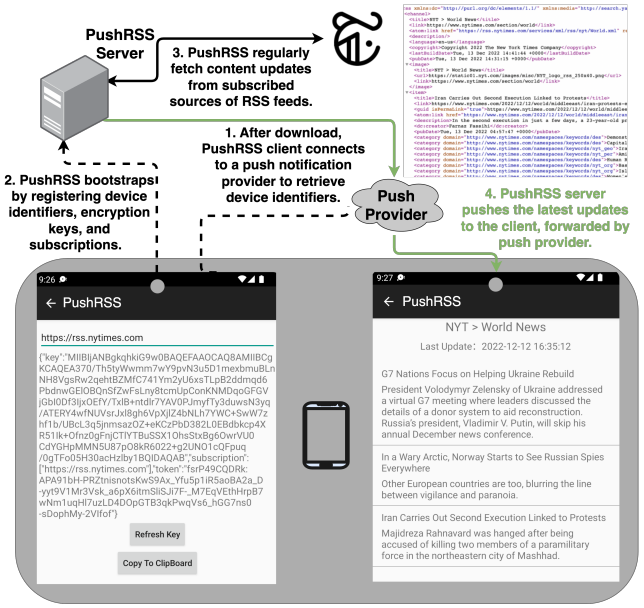

Figure 2: **Workflow of *PushRSS*** After the initial registration, *PushRSS* server continues to push updates to the client through a push notification service.

the latest contents on their behalf, and then delivers all updates to the user in a single location for easy access. The traffic pattern generated by an RSS aggregator is highly asymmetric: only a one-time registration and feed subscription need to be sent upstream, while downstream traffic can be fairly sizable depending on the number of sources subscribed to and the size and frequency of updates. This traffic pattern aligns well with the Publish/Subscribe model of push notification services.

Unlike traditional RSS aggregators that rely on direct connections between a client app and a server, *PushRSS* tunnels downstream traffic carrying regular content updates through a push notification provider's network. Users are only required to perform a one-time bootstrapping step, registering their device tokens and subscription lists. *PushRSS* then continuously fetches updates and delivers them using push notification as a transport. As it doesn't require a direct connection to its server, once bootstrapped, *PushRSS* can continue to operate even if the server IP is blocked.

3.1 Design

Figure 2 provides an overview of *PushRSS*'s workflow. After download, *PushRSS* client contacts a push provider (e.g. APNS, FCM, etc.) to register the device, retrieve device tokens, and send them to the *PushRSS* server along with a list of subscribed publishers providing feeds compatible with RSS 2.0 format [37]. The *PushRSS* server then regularly fetches updates from the subscribed publishers, parses the contents into structured formats, and then delivers to the client by embedding the contents inside the "*data*" field of push notification requests, indexed by the client's identifiers.

Bootstrapping When the *PushRSS* app is launched for the first time, it registers the app's instance with a push notification provider to obtain device- and app-specific identifiers, typically in the form of tokens. *PushRSS* client then attempts to send the device identifiers, public keys to encrypt update payloads, and user-specified subscription list directly to the *PushRSS* server. However, if direct communication to the server is blocked in a censored region, the user may instead use out-of-band channels, such as domain fronting, email services, or encrypted instant messaging apps, to send this information. For Android devices, another option to send upstream registration info is to use FCM's "upstream push notifications," which allows the client to send push messages back to the app server via the XMPP protocol [17].

Encryption and Reliable Delivery Most push notification services do not offer built-in end-to-end encryption. For example, Google FCM uses point-to-point encryption with two separate TLS connections connecting the FCM server to the app server and client device. To achieve end-to-end encryption, the *PushRSS* client registers its public key with the *PushRSS* server at bootstrapping time. For each subsequent push notification update, the payload is encrypted with an ephemeral, message-specific key, which is then encrypted with the client's public key and added to the message. The final push notification payload includes the encrypted update and a signature to ensure integrity. External solutions, such as Capillary [23], exist to simplify the process of sending end-to-end encrypted push notification messages.

A reliable delivery mechanism is required to transmit update contents that exceed the maximum payload size of a single push notification message (4 KiB for APNS/FCM). To address this, we draw inspiration from previous work on VoIP-based circumvention systems that use XOR-based encoder/decoder to ameliorate potential packet loss [43]. Specifically, we can transmit a redundant push notification message after every N messages, which is constructed by XOR-ing the previous N messages together. This ensures that the original content can be recovered as long as N out of $N+1$ messages from the same group are received by the client. The value of N is determined based on network conditions and the choice of push notification provider.

3.2 Applications

PushRSS facilitates the delivery of contents when their publishers and users are separated by firewalls. *PushRSS* only distributes contents based on subscriptions and does not allow users to request contents on-demand due to the lack of a persistent upstream channel. Despite this limitation, there are still several applications that can benefit from this type of communication pattern. For example, a user living in a country with strict censorship on news and media can use *PushRSS* to receive a continuous feed from global news agencies and personal blogs. In these situations, the network traffic

between the user and the content source is highly asymmetric: the upstream channel is only required at bootstrapping time, while the downstream channel needs to be censorship-resistant, reliable over time, and have acceptable bandwidth. We note that all major news publishers offer standard RSS feeds categorized by region, sector, and interest. Additionally, platforms such as Medium.com and tools like WordPress RSS plugin make it easy for individual publishers to publish their contents in RSS-compatible formats.

Additionally, *PushRSS* can facilitate the bootstrapping stage for existing circumvention systems. Sometimes referred to as the *Identifier Distribution Problem*, bootstrapping is often the weakest link in many circumvention systems and has been studied in previous research [18, 31, 40, 44]. Many circumvention systems require pre-shared secrets for bootstrapping, such as server IP, port, obfuscation key, *etc.*, which need to be regularly updated on both sides, for example, when rotating to a new IP/port after a blocking event. To address this, some circumvention client apps offer a "subscription" feature to replace static keys, allowing server operators to publish and update configurations on a URL that is constantly being polled by the client app. Integrating *PushRSS*'s publish/subscribe model into existing circumvention systems may further optimize this process by changing the "polling" process into a "pushing" process: instead of clients repeatedly polling the URL where the configuration is stored, the server can automatically push updates to the client whenever a subscribed configuration is changed. This approach not only improves efficiency, but it also enhances resilience against blocking through the use of push notification services.[2] This proposed identifier distribution approach is illustrated in Appendix Figure 9.

4 *PushProxy*: Asymmetric Proxy Based on Push Notification Services

We design and implement *PushProxy*. At a high level, *PushProxy* sends users' downstream traffic through a push notification service (Google FCM [15] in our case), while keeping upstream an independent/direct channel. We first provides an overview of the proxy system in § 4.1. Then, we describe a few design details in § 4.2 that enhance the usability despite the inherent constraints of push notification services, such as rate limiting and the lack of reliability.

The key advantage of *PushProxy* over traditional symmetric proxies comes from its ability to decouple the upstream and downstream traffic. Such decoupling, sometimes referred to as "triangular routing" [36, 43], mitigates the ability of a network adversary to perform website fingerprinting or deep packet inspection (DPI) for censorship at the per-flow level

[2]For example, OutlineVPN offers a subscription feature called "dynamic key" [14] and recommends to store configurations on pad.riseup.net. However, the site itself is blocked in many regions including China.

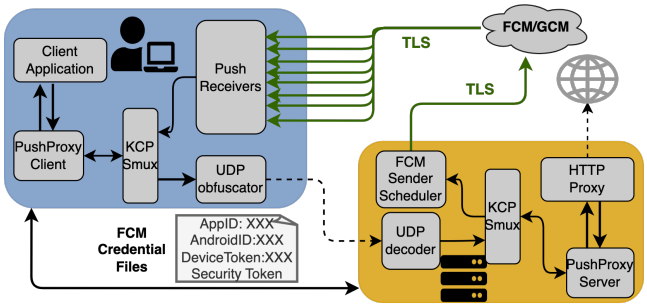

Figure 3: ***PushProxy* Overall Diagram.** *PushProxy* routes downstream traffic through a push notification service, while keeping upstream an independent channel. In our implementation, we used FCM for downstream and XOR-obfuscated UDP for stream.

(i.e., based on 3-tuple or 4-tuple, but not on the aggregate traffic a client sends/receives). For example, previous work on attacking obfuscated circumvention tools using traffic analysis often assume an adversary with symmetric visibility and is able to correlate a flow's upstream traffic with its downstream traffic [8, 21, 28, 41, 48]. Triangular routing makes such correlation more challenging, especially when different transport protocols are used in each direction. While triangular routing itself does not obfuscate the underlying traffic, it can be complementary to existing obfuscation strategies by further limiting the amount of information an adversary can learn or correlate. Compared to other triangular routing designs such as CensorSpoofer and ReQrypt [36, 43], *PushProxy* prevails due to its use of service tunneling to hide the downstream traffic inside legitimate, persistent connections to push notification servers. This approach eliminates the need for IP spoofing and results in improved blocking resistance and better compatibility with NAT. Additionally, push notification services have lower latency and higher bandwidth compared to other channels used in existing designs (e.g. VoIP), leading to an improvement of performance (see § 5).

4.1 Design

Figure 3 illustrates the components of a *PushProxy* system. The *PushProxy* client listens on a local address for incoming connections and the *PushProxy* server forwards the data stream to an HTTP proxy on the server side. Both the client and the server are implemented in Golang.

Bootstrapping Users of *PushProxy* need to complete a bootstrapping stage before they can use the tool. Similar to symmetric, authenticated proxies (e.g. obfs4, shadowsocks), *PushProxy* assumes that some information are shared between the client and the server prior to usage, including server identifiers (upstream listening address and FCM *senderID* [15]) and client identifiers (FCM's *deviceToken*). All client identifiers are generated by registering with FCM, which can be done either at client side or server side. An optional *ClientID* can be specified to support a multi-client setting where a sin-

gle instance of *PushProxy* server is shared between multiple *PushProxy* clients.

Downstream Downstream data is base64-encoded and included in the "*data*" field of FCM messages, which are sent by Firebase Go SDK and received from persistent connections that the client maintains with FCM endpoints. Notification (user prompt) for these data-carrying push messages is disabled. We set the *time-to-live* field to zero in order to avoid throttling by FCM [3], which essentially makes downstream a best-effort channel. We do not define additional fields for traffic control (e.g. SEQ, ACK) but instead rely on the reliability provided by upper layer protocols (See § 4.2).

Upstream Our proof-of-concept uses XOR-obfuscated UDP packets to carry upstream traffic, similar to the XOR patch [34] developed to obfuscate OpenVPN traffic. However, *PushProxy* is designed to be flexible regarding how upstream traffic is handled and in no way depends on specific upstream transport to function. We believe *PushProxy*'s asymmetric routing potentially allows for more flexible obfuscator design, as for each direction, half of the traffic is dummy data that could be arbitrarily interleaved with the uni-directional application streams.

4.2 Reliability, Multiplexing, Encryption

Since the delivery of push notification messages is a best-effort process, we need to introduce an additional sequencing mechanism to provide a reliable interface to tunnel user stream over the potentially unreliable push notification transport. Several options are available from previous work, such as to use forward error correction algorithms (e.g. [43]) or to implement custom reliability schemes (e.g. [39]).

Instead, *PushProxy* employs a *TurboTunnel* design by separating an abstract reliability layer from the underlying circumvention transports [19]. Figure 10 illustrates this design, where a KCP session is built on top of the obfuscation layer using UDP or push notification for different directions. The KCP session is tuned to optimize for the push notification provider being used. For example, downstream MTU in our implementation is configured as the maximum payload size of an FCM message with framing overhead subtracted. Multiplexing support is added by smux [38]. Encryption at the KCP layer is disabled and delegated to the application layer.

4.3 Rate Limit

According to FCM documentation, there is a per-destination rate limit for sending push notifications [15]. The rate limit is set to 5000 messages per hour, which when combined with the maximum payload size of 4 KiB gives a sub-optimal bandwidth of 40kbps. While occasional bursts of traffic are allowed (we were able to get a maximum throughput of 5mbps before push notification messages are getting dropped), we believe staying below rate limit would provide more stable proxy connections for users.

[3] https://firebase.google.com/docs/cloud-messaging/concept-options#ttl

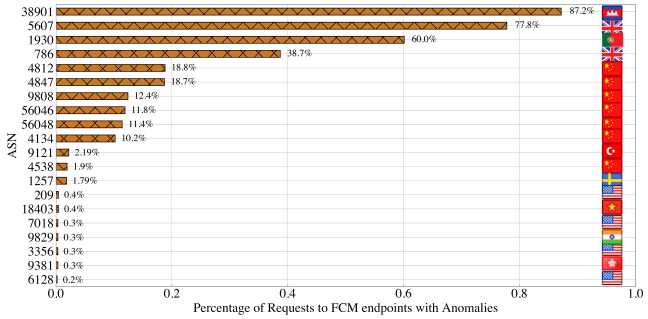

Figure 4: ***Hyperquack* Measurement Results** - We run *Hyperquack* for 7 months and aggregate connection anomalies at AS level.

We notice that the rate limit is specified per *deviceToken*. Therefore, instead of maintaining one connection between *PushProxy* client and FCM, we register N client instances and spawn N persistent connections to FCM, each parameterized by a unique *deviceToken*. *PushProxy* server sends push notification using N *deviceTokens* in a round-robin manner, while *PushProxy* client simultaneously listens with N push receivers and processes the data as soon as one becomes ready. Rate limits are enforced by specifying a maximum window size (in number of packets in flight) for the KCP session. The maximum window size is calculated as $N * MaxRate/s * RTT$, where RTT is the Round Trip Time and N is the number of available *deviceTokens* to send push notifications to. In § 5, we show that the downstream push notification channel can be scaled to support general browsing activities while staying under the rate limit specified by the service provider.

5 Evaluation

We assess the viability of using push notifications as a transport for circumvention traffic, considering both its availability across regions and over time, as well as its performance in comparison with other circumvention tools.

5.1 Availability

To use push notification services for effective censorship circumvention, it is crucial that these services are not subject to blocking themselves, particularly in regions where censorship is prevalent. We conduct measurements to test for potential censorship practices against push notification services across regions and networks. To be consistent with our prototype implementation, we focus on Google FCM, one of the largest push notification providers worldwide. We aim to determine 1) *Which networks (ASes) actively block connections to FCM services?* And 2) *Does FCM offer high availability by remaining reachable over time?*

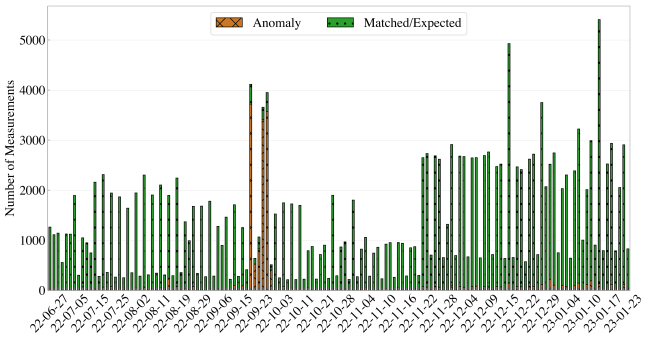

Figure 5: **Longitudinal *Hyperquack* results for China.**

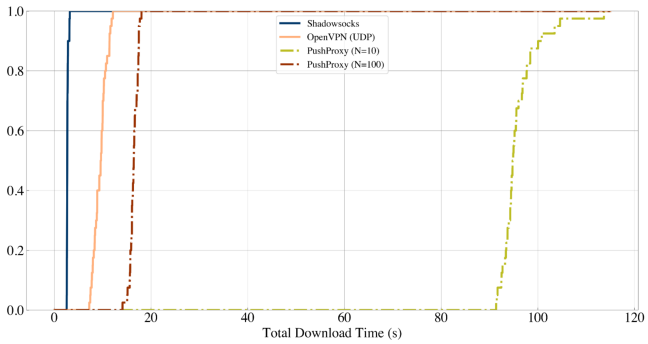

Figure 6: **Bandwidth** The median total download time is 2.70s for Shadowsocks, 9.68s for OpenVPN (UDP), 94.92s for *PushProxy* ($N = 10$), and 16.46s for *PushProxy* ($N = 100$). All tests were conducted at 3:00 AM Beijing Time (GMT+8)

Methodology We used peer-reviewed technique Hyperquack [35] to measure potential HTTPS blocking against connections to FCM endpoints. Hyperquack measures keyword blocking at the application layer using web servers located in a targeted region/network. It first builds up a template of expected behaviors for a targeted web server. Next, it sends HTTPS requests with targeted SNIs and monitors for potential signs of blocking (e.g., TCP reset, blockpage, timeout *etc.*).

We target FCM endpoints found in the official documentation [3], included in Appendix Table 1. The measurement commenced on June 27, 2022 and continued until January 24, 2023. During this period, we conducted four rounds of Hyperquack measurements each week.

Results We collected a total of 5,555,298 measurements, targeting web servers located in 1,632 unique ASes. We aggregated results at the AS level, removing servers whose behaviors showed a large discrepancy from other servers in their network and only reporting on ASes where we had a sufficient number of targets (> 20). Figure 4 shows the aggregated results. We found that only a small number of ASes actively interfere with HTTPS connections to FCM endpoints. Among the ASes with the highest percentage of anomalies, AS5607, AS1930, and AS786 exhibited aggressive blocking behaviors that were dissimilar to any other ASes in their respective countries. Since anomalies were also measured in these networks for other Google-related services and many other "benign" domains, we believe it is unlikely that the anomalies we observed suggest targeted censorship against FCM services.

China stands out as a notable exception, as there is a certain level of consistency in the interference observed across ASes, indicating potential nation-wide, coordinated efforts to block FCM. Longitudinal analysis of measurements conducted on web servers located in China (Figure 5) reveals that the majority of blocking events occurred between September 22 and September 30, 2022, during which almost all measurements failed with TCP reset. This is corroborated by user reports online of other Google services being unavailable during the same time period [16]. On the eve of the Party's National Congress, the most sensitive political event in the country, China's online censor tightened its grip by aggressively blocking all subdomains of the form **.google.com*[4]. The blocking was later lifted for FCM endpoints on October 1. However, it is important to note that the censor did not simply reverse the blocking rule for **.google.com*, as many subdomains that were blocked on September 22, including {docs/groups/sites}.google.com, continue to remain blocked as of February 2023. The lifting of the blocking specifically for FCM suggests that an exception was made due to the disruption caused to services and apps that rely on FCM for their functionality. This further highlights that push notification, as a circumvention transport, benefits from the high collateral damage that a censor has to sustain by blocking it.

5.2 Performance

We evaluate the performance of push notification as a downstream service-tunneling channel by comparing *PushProxy* with other circumvention tools in a realistic setting. We choose OpenVPN and Shadowsocks for comparison.

Methodology To emulate a realistic setting, we deployed a *PushProxy* server, an OpenVPN Access Server (v2.10.3), and a Shadowsocks server (*go-shadowsocks2* v0.1.5) and co-located them on a measurement machine located in Michigan, USA. Next, we launched their corresponding clients from a premium VM in Shanghai, China, and had the clients fetch resources from a web server located in Virginia, USA through one of the three proxying channels. We confirmed that the client VM is located inside China with traceroutes and latency measurements, and we also observed DNS and SNI-based censorship behaviors similar to the ones previously noted in studies on the GFW. The round trip time is around 219ms between the China clients and the proxy servers, and 25ms between the proxies and the web server.

Bandwidth To evaluate bandwidth, we performed 40 measurements of the total time taken through different proxies to

[4]Note that most of Google services have been blocked in China for years. However, a handful of less-sensitive services, such as FCM, Google Docs, and Google Translate, remained available before this blocking.

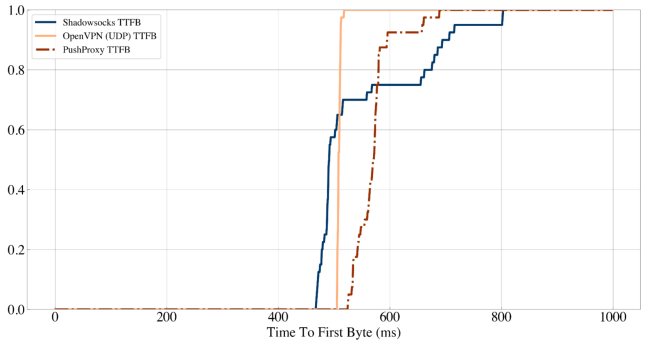

Figure 7: **Time To First Byte** - From 40 measurements, we found the median time to first byte was 492ms for Shadowsocks, 508ms for OpenVPN (UDP), and 572ms for *PushProxy*.

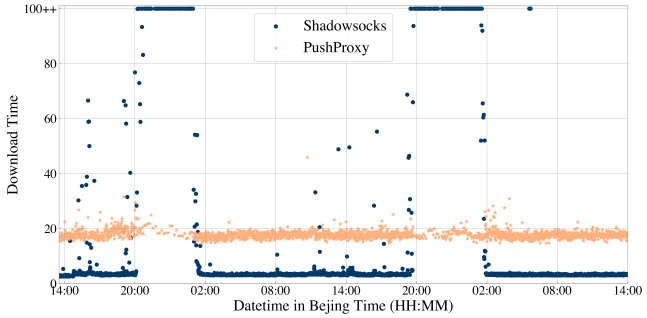

Figure 8: **Bandwidth Diurnal Pattern** - The performance of *PushProxy* remains stable during peak hours.

fetch a 10-megabyte file from the web server. As shown in Figure 6, traditional symmetric proxies provide better bandwidth, resulting in median download time of 2.70s and 9.68s for Shadowsocks and OpenVPN, respectively. However, we note that despite being under the constraints of rate limiting, adding paralleled push notification receivers is an effective way to scale up the downstream bandwidth. By increasing the number of push receivers (N) from 10 to 100, the median download time is reduced from 94.92s to 16.46s, without breaking FCM's rate limit. This gives us a bandwidth of around 4.86 Mbps, which is significantly higher than other service tunneling systems such as dnstt [2] (1.5 Mbps) or CensorSpoofer [43] (64 Kbps) and is able to support general browsing activities.

Time To First Byte (TTFB) We connected the clients to proxies to request resources from the web server. We performed 40 such measurements for *PushProxy* ($N = 100$), Shadowsocks, and OpenVPN (UDP mode). Results are shown in Figure 7. The median TTFB for Shadowsocks and OpenVPN is 492ms and 508ms, respectively, and the TTFB for *PushProxy* is 572ms.

Diurnal Patterns During the evaluation, we conducted multiple rounds of bandwidth measurements and observed intermittent network slowdowns that resembled network congestion, with increased packet losses and decreased throughput. Further testing indicated that these slowdowns followed a diurnal pattern. Figure 8 shows a 48-hour period during which we continuously performed the bandwidth measurements using Shadowsocks and *PushProxy*. The graph clearly suggests that for Shadowsocks, performance starts to degrade from 20:00 Beijing time (GMT+8) and only recovers after 2:00. During these peak hours, the hourly-averaged downloading time for the same file increased from 3 seconds to over 100 seconds. Direct connection (without using any proxy) to the web server results in similar slowdown. While we do not have ground truth for why the slowdown is happening, previous research examining transnational Internet performance has noted their measurement clients in China suffer from occasional slowdowns following a similar diurnal pattern, which the authors hypothesized financial motivations to be the cause [50].

Interestingly, we found that the performance of *PushProxy* remained stable and exhibit acceptable throughput even during these peak hours. We suspect the reason to be *PushProxy*'s use of service tunneling in downstream direction through FCM, whose endpoints domain names resolves to a domestic IP from our China VPS. Previous work has found that websites with servers physically located inside China do not suffer from the diurnal slowdown pattern [50]. This suggests that *PushProxy*, despite its use of triangular routing, may even provide better user experience than symmetric proxies under certain situations, such as web browsing where a higher downstream bandwidth during peak hours can make a significant difference in the quality of the user's experience.

6 Discussion

6.1 Publish/Subscribe Model of Circumvention

PushRSS strictly conforms to a Publish/Subscribe model, in which users performs a one-time subscription for a publisher's feed, and *PushRSS* is responsible for delivering subsequent content updates to all subscribers of the feed. While this approach offers several advantages (§ 2), the usefulness of the tool is contingent on the availability of various feeds. This publisher-centric approach is similar to that utilized by CacheBrowser [24]. Although in this case publishers must invest time and resources in presenting their contents in structured formats, we believe they are likely motivated to do so in order to increase the accessibility to their contents.

Many publishers, including major news outlets such as NYT and BBC, already offer standard RSS feeds for their content, which are often organized by region and topic. However, we found that these feeds typically only include a title and a brief summary of each item, requiring users to click through to the original source for the full story. The presence of paywalls and third-party resources (e.g., images) can further detract from the user experience. We encourage publishers to use more expressive structures that provide enough contents to reconstruct significant portions of a webpage and

offer a reading experience akin to traditional web browsing. Alternatives to RSS include Telegram's Instant Views [5] and the BIFROST-T service [1], which parse webpages and WordPress posts into structured formats.

6.2 Limitation

Traffic Analysis *PushRSS* and *PushProxy* do not protect against fingerprinting attacks based on traffic analysis. A censor may be able to distinguish push notification flows carrying circumvention traffic based on features such as packet length and timing distribution. While there is no current evidence of sophisticated traffic analysis deployment by real-world censors, simple rule-based filters could potentially suffice, as high-frequency, high-bandwidth push notifications are quite atypical from standard push notification usage and can be easy for censors to identify and throttle. (e.g., it takes over 100 push notifications to load a typical website, contrasting notably with the daily average of 46 push notifications received by a smartphone [4].)

Another potential vector of attack enabled by analyzing traffic of *PushRSS* is the de-anonymization of subscribers to specific content feed. More specifically, an adversary subscribing to a notification feed could potentially identify other subscribers of the same feed by correlating their traffic characteristics, using techniques based on packet timings and sizes [7, 9]. In this case, the goal of the adversary is not to disrupt connectivity, but rather to ascertain that a user has installed an anti-censorship tool or subscribed to a feed containing sensitive contents. Such risks need to be recognized and investigated. Potential mitigation strategies include introducing randomness in the delivery timing of notifications or restricting access only to trusted users.

Platform Censorship Push notification providers that consciously support censorship circumvention may face pressure from the censor, who can threaten to block or take legal action. The censor may demand the provider to disable push notifications for circumvention apps or reduce the bandwidth by lowering the rate limit, rendering push notification as a transport less useful. A broader adoption of push notifications for circumvention purposes is likely to motivate censors to impose stricter technical and policy controls over such communication channels. However, we note that despite the majority of Google services being blocked in China since 2014, push notifications powered by FCM remain accessible as of June 2023. This observation may hint at the economic and societal repercussions that would arise from blocking such a service, possibly creating a backlash that outweighs the benefits of censorship.

7 Related Work

Censorship circumvention necessitates traffic obfuscation to evade detection and blocking by censors. Most obfuscation strategies fall into three categories: mimicry, randomization, or service-tunneling. Mimicry-based obfuscations [13, 32, 45] work by simulating traffic characteristics of "benign" protocols. Previous work have since found several attacks that result in effective protocol fingerprinting, as seamlessly mimicking another protocol is extremely challenging [25, 41]. On the other hand, randomization-based strategies aim to eliminate all fingerprints by encrypting traffic into bits indistinguishable from random. However, being random could itself becomes a feature and censors could block fully encrypted traffic during politically sensitive times [47].

A logical extreme of mimicry, named *service tunneling*, is to tunnel circumvention traffic over actual, public-facing network services. Examples of this approach include CloudTransport, SWEET, FreeWave, and dnstt, which relay circumvention traffic via cloud storage, email, VoIP, and DNS, respectively [2, 10, 26, 27]. *PushProxy* falls under this category, as it tunnels downstream circumvention traffic through existing, persistent connections between end user devices and push notification servers. Service tunneling provides better blocking resistance in that the address of the proxy server is effectively hidden from the censor – the censor only sees the address of the public-facing service. Blocking such services outright would result in significant collateral damage and is therefore unlikely to be implemented by the censor.

An additional feature of *PushProxy* is the decoupling of upstream and downstream traffic. Such decoupling provides some level of defense against traffic analysis by reducing information exposed to attackers with a limited network perspective [12, 42]. While correlation between the upstream and downstream flows is still possible, network devices primarily use the four tuple for connection identification, making correlation challenging and likely causing false positives/negatives. Using different transport protocols for upstream and downstream introduces additional complexity. *PushProxy* is similar to CensorSpoofer [43] in that they both decouple upstream and downstream traffic. However, *PushProxy* provides better performance, as shown in § 5, due to its higher bandwidth from push notification services compared to VoIP.

8 Conclusion

We explore the potential of using push notification for censorship circumvention. Designed for time-sensitive tasks, push notifications provide a downstream-only, blocking-resistant data transport with low-latency, mid-bandwidth at minimal cost. Empirical evidence suggests that adversaries are wary of censoring push notification services due to the potential for economic and social collateral damage. We present two tools, *PushRSS* and *PushProxy*, that leverage this property to tunnel downstream circumvention traffic. While these tools have their limitations, we believe that push notification still shows promise as a viable circumvention transport that complements existing approaches.

9 Acknowledgment

The authors are grateful to the anonymous reviewers for their constructive feedback. This material is based upon work supported by the National Science Foundation under Grant No.2237552, No.2141512, and the Defense Advanced Research Projects Agency (DARPA) under Agreement HR00112190127.

References

[1] Circumvention of website blocking by means of telegram. https://www.qurium.org/bifrost-t/.

[2] DNS tunnel over DNS over HTTPS (DoH) or DNS over TLS (DoT) resolvers. https://www.bamsoftware.com/software/dnstt/.

[3] FCM ports and your firewall. https://firebase.google.com/docs/cloud-messaging/concept-options#messaging-ports-and-your-firewall.

[4] Push Notification Statistics (2021). https://www.businessofapps.com/marketplace/push-notifications/research/push-notifications-statistics/.

[5] Telegram: Instant Views Explained. https://instantview.telegram.org/.

[6] Alice, Bob, Carol, J. Beznazwy, and A. Houmansadr. How China Detects and Blocks Shadowsocks. In *ACM Internet Measurement Conference (IMC)*, 2020.

[7] A. Back, U. Möller, and A. Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In *Information Hiding: 4th International Workshop, IH 2001 Pittsburgh, PA, USA, April 25–27, 2001 Proceedings*, pages 245–257. Springer, 2001.

[8] D. Barradas, N. Santos, and L. Rodrigues. Effective Detection of Multimedia Protocol Tunneling Using Machine Learning. SEC'18, USA, 2018. USENIX Association.

[9] A. Bozorgi, A. Bahramali, F. Rezaei, A. Ghafari, A. Houmansadr, R. Soltani, D. Goeckel, and D. Towsley. I Still Know What You Did Last Summer: Inferring Sensitive User Activities on Messaging Applications Through Traffic Analysis. *IEEE Transactions on Dependable and Secure Computing*, 2022.

[10] C. Brubaker, A. Houmansadr, and V. Shmatikov. CloudTransport: Using Cloud Storage for Censorship-Resistant Networking. volume 8555, pages 1–20, 07 2014.

[11] Y. Chen, T. Li, X. Wang, K. Chen, and X. Han. Perplexed Messengers from the Cloud: Automated Security Analysis of Push-Messaging Integrations. CCS '15, page 1260–1272, New York, NY, USA, 2015. Association for Computing Machinery.

[12] W. De la Cadena, A. Mitseva, J. Hiller, J. Pennekamp, S. Reuter, J. Filter, T. Engel, K. Wehrle, and A. Panchenko. TrafficSliver: Fighting Website Fingerprinting Attacks with Traffic Splitting. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1971–1985, New York, NY, USA, 2020. Association for Computing Machinery.

[13] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. Protocol Misidentification Made Easy with Format-Transforming Encryption. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer Communications Security*, CCS '13, page 61–72, New York, NY, USA, 2013. Association for Computing Machinery.

[14] OutlineVPN: Dynamic Access Keys. https://www.reddit.com/r/outlinevpn/wiki/index/dynamic_access_keys/.

[15] About FCM messages. https://firebase.google.com/docs/cloud-messaging/concept-options.

[16] The Great Firewall of China has blocked google.com and all its subdomains. https://github.com/net4people/bbs/issues/128.

[17] FCM: Sending upstream messages on Android. https://firebase.google.com/docs/cloud-messaging/android/upstream.

[18] N. Feamster, M. Balazinska, W. Wang, H. Balakrishnan, and D. Karger. Thwarting Web Censorship with Untrusted Messenger Discovery. In *Privacy Enhancing Technologies*, pages 125–140. Springer, 2003.

[19] D. Fifield. Turbo Tunnel, a good way to design censorship circumvention protocols. In *Free and Open Communications on the Internet*. USENIX, 2020.

[20] D. Fifield, C. Lan, R. Hynes, P. Wegmann, and V. Paxson. Blocking-Resistant Communication through Domain Fronting. *Proceedings on Privacy Enhancing Technologies*, 2015, 06 2015.

[21] My Experience With the Great Firewall of China. https://blog.zorinaq.com/my-experience-with-the-great-firewall-of-china/.

[22] Y. Han, D. Xu, J. Gao, and L. Zhu. Using blockchains for censorship-resistant bootstrapping in anonymity networks. In C. Alcaraz, L. Chen, S. Li, and P. Samarati, editors, *Information and Communications Security*, pages 240–260, Cham, 2022. Springer International Publishing.

[23] G. Hogben and M. Perera. Capillary. https://github.com/google/capillary, 2018.

[24] J. Holowczak and A. Houmansadr. CacheBrowser: Bypassing Chinese Censorship without Proxies Using Cached Content. CCS '15, page 70–83, New York, NY, USA, 2015. Association for Computing Machinery.

[25] A. Houmansadr, C. Brubaker, and V. Shmatikov. The Parrot Is Dead: Observing Unobservable Network Communications. In *2013 IEEE S&P*.

[26] A. Houmansadr, T. J. Riedl, N. Borisov, and A. C. Singer. I want my voice to be heard: IP over Voice-over-IP for unobservable censorship circumvention. In *Network and Distributed System Security Symposium*, 2013.

[27] A. Houmansadr, W. Zhou, M. Caesar, and N. Borisov. SWEET: Serving the Web by Exploiting Email Tunnels. *IEEE/ACM Transactions on Networking*, 25(3):1517–1527, 2017.

[28] C. Kwan, P. Janiszewski, S. Qiu, C. Wang, and C. Bocovich. Exploring Simple Detection Techniques for DNS-over-HTTPS Tunnels. FOCI '21, page 37–42, New York, NY, USA, 2021. Association for Computing Machinery.

[29] H. Lee, T. Kang, S. Lee, J. Kim, and Y. Kim. Punobot: Mobile botnet using push notification service in android. In Y. Kim, H. Lee, and A. Perrig, editors, *Information Security Applications*, pages 124–137, Cham, 2014. Springer International Publishing.

[30] T. Li, X. Zhou, L. Xing, Y. Lee, M. Naveed, X. Wang, and X. Han. Mayhem in the Push Clouds: Understanding and Mitigating Security Hazards in Mobile Push-Messaging Services. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, page 978–989, New York, NY, USA, 2014. Association for Computing Machinery.

[31] P. Lincoln, I. Mason, P. Porras, V. Yegneswaran, Z. Weinberg, J. Massar, W. Simpson, P. Vixie, and D. Boneh. Bootstrapping Communications into an Anti-Censorship System. In *Free and Open Communications on the Internet*. USENIX, 2012.

[32] H. Mohajeri Moghaddam, B. Li, M. Derakhshani, and I. Goldberg. SkypeMorph: Protocol Obfuscation for Tor Bridges. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, page 97–108, New York, NY, USA, 2012. Association for Computing Machinery.

[33] Learning more about the GFW's active probing system. https://blog.torproject.org/learning-more-about-gfws-active-probing-system.

[34] OpenVPN_XORPatch. https://github.com/clayface/openvpn_xorpatch.

[35] R. S. Raman, P. Shenoy, K. Kohls, and R. Ensafi. Censored Planet: An Internet-wide, Longitudinal Censorship Observatory. In *Computer and Communications Security*. ACM, 2020.

[36] ReQrypt: A censorship circumvention tool. https://github.com/basil00/reqrypt.

[37] RSS 2.0 Specification. https://support.google.com/merchants/answer/160589?hl=en.

[38] smux: A Stream Multiplexing Library for golang with least memory usage(TDMA). https://github.com/xtaci/smux.

[39] Survey of techniques to encode data in DNS messages. https://www.bamsoftware.com/software/dnstt/survey.html.

[40] M. C. Tschantz, S. Afroz, Anonymous, and V. Paxson. SoK: Towards Grounding Censorship Circumvention in Empiricism. In *2016 IEEE Symposium on Security and Privacy (SP)*, 2016.

[41] L. Wang, K. P. Dyer, A. Akella, T. Ristenpart, and T. Shrimpton. Seeing through Network-Protocol Obfuscation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 57–69, New York, NY, USA, 2015. Association for Computing Machinery.

[42] M. Wang, A. Kulshrestha, L. Wang, and P. Mittal. Leveraging strategic connection migration-powered traffic splitting for privacy, 2022.

[43] Q. Wang, X. Gong, G. T. Nguyen, A. Houmansadr, and N. Borisov. CensorSpoofer: Asymmetric Communication Using IP Spoofing for Censorship-Resistant Web Browsing. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, page 121–132, New York, NY, USA, 2012. Association for Computing Machinery.

[44] Q. Wang, Z. Lin, N. Borisov, and N. J. Hopper. rBridge: User Reputation based Tor Bridge Distribution with Privacy Preservation. In *Network and Distributed System Security*. The Internet Society, 2013.

[45] Z. Weinberg, J. Wang, V. Yegneswaran, L. Briesemeister, S. Cheung, F. Wang, and D. Boneh. StegoTorus: A Camouflage Proxy for the Tor Anonymity System. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, page 109–120, New York, NY, USA, 2012. Association for Computing Machinery.

[46] Wikipedia. Observer pattern — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Observer%20pattern&oldid=1135085404, 2023. [Online; accessed 23-January-2023].

[47] M. Wu, J. Sippe, D. Sivakumar, J. Burg, P. Anderson, X. Wang, K. Bock, A. Houmansadr, D. Levin, and E. Wustrow. How the Great Firewall of China Detects and Blocks Fully Encrypted Traffic. In *32th USENIX Security Symposium (USENIX Security 23)*.

[48] D. Xue, R. Ramesh, A. Jain, M. Kallitsis, J. A. Halderman, J. R. Crandall, and R. Ensafi. OpenVPN is Open to VPN Fingerprinting. In *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA, 2022. USENIX Association.

[49] S. Zhao, P. P. C. Lee, J. C. S. Lui, X. Guan, X. Ma, and J. Tao. Cloud-Based Push-Styled Mobile Botnets: A Case Study of Exploiting the Cloud to Device Mes-

FCM endpoints used in Hyperquack measurement
mtalk.google.com
mtalk4.google.com
mtalk-staging.google.com
mtalk-dev.google.com
alt1-mtalk.google.com
alt2-mtalk.google.com
alt3-mtalk.google.com
alt4-mtalk.google.com
alt5-mtalk.google.com
alt6-mtalk.google.com
alt7-mtalk.google.com
alt8-mtalk.google.com

Table 1: **FCM Endpoints used in Hyperquack measurement**. The set of endpoints were collected from the FCM documentation [3].

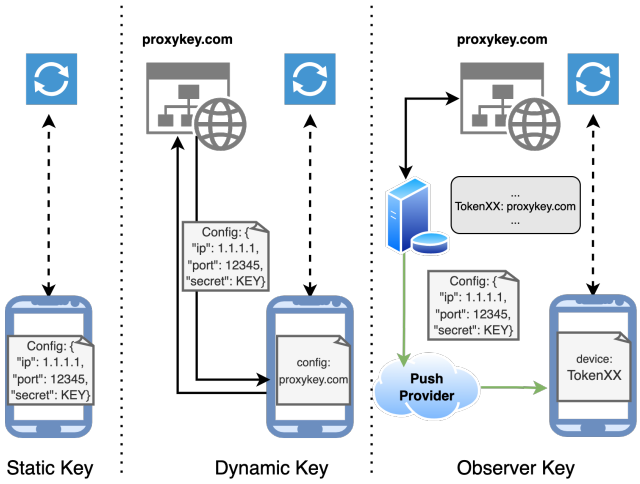

Figure 9: **Different Identity Distribution Mechanisms.** Observer Key [46] changes the "polling" process from the dynamic key into a "pushing" process by having the server automatically sends updates to the client whenever a subscribed configuration is changed.

saging Service. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, page 119–128, New York, NY, USA, 2012. Association for Computing Machinery.

[50] P. Zhu, K. Man, Z. Wang, Z. Qian, R. Ensafi, J. Halderman, and H. Duan. Characterizing Transnational Internet Performance and the Great Bottleneck of China. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 4:1–23, 05 2020.

A Appendix

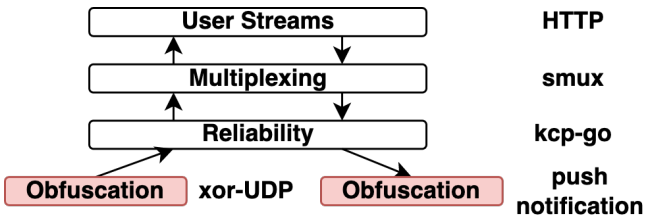

Figure 10: **The TurboTunnel Design of *PushProxy*.** TurboTunnel [19] decouples reliability and obfuscation.