# Looking at the Clouds: Leveraging Pub/Sub Cloud Services for Censorship-Resistant Rendezvous Channels

Afonso Vilalonga
Universidade NOVA de Lisboa &
NOVA LINCS
Portugal
j.vilalonga@campus.fct.unl.pt

João S. Resende
Universidade do Porto
Portugal
jresende@fc.up.pt

Henrique Domingos
Universidade NOVA de Lisboa &
NOVA LINCS
Portugal
hj@fct.unl.pt

## ABSTRACT

Many censorship evasion systems rely on establishing a connection between the user and a proxy that acts as the gateway to censored content. However, informing the user about proxy addresses or exchanging the necessary information to establish a connection between the user and the proxy when the user resides in a censored region without access to non-blocked proxies is not a trivial task. In this paper, we address the problem of creating a censorship-resistant communication channel where information about how to establish these user-proxy connections (e.g., proxy IPs) can be effectively transmitted with a low risk of a censor blocking the communication channel, even if the censor has the same knowledge of how to operate them as the user. To this end, we designed and developed a prototype of a rendezvous protocol — a censorship-resistant communication channel for data transmission typically used for bootstrapping connections in censorship evasion systems — leveraging Pub/Sub cloud services, a popular and widely used service available across different cloud providers.

## KEYWORDS

Anti-censorship, Rendezvous protocols, Cloud services, Communication protocols

## 1 INTRODUCTION

The Internet's potential for global connectivity and dissemination of information makes it a primary target for censorship by authoritarian regimes (i.e., censors) [4, 10, 27, 28, 37]. These regimes perceive unrestricted access to information as a threat and, as such, aim to assert control over their population, manipulate information, and maintain the existing status quo within their jurisdiction by enforcing online censorship [21, 22]. In response to these challenges, censorship evasion systems have emerged to enable users to maintain connections while traversing censored regions. Censorship evasion systems employ various strategies to remain unblocked, such as tunneling data in "regular" protocols [3, 5, 31], protocol mimicking [25], traffic obfuscation [1], steganography [13], etc. Typically, these systems operate by having the user establish a connection with a proxy using one of the above methodologies, which then routes the user's traffic to censored content [32]. Throughout this paper, we adopt the term "Bridge" from Tor [7] to denote any proxy, server, or network node in a censorship evasion system. However, using censorship evasion systems in heavily censored regions is often restricted, and the IPs of bridges are frequently blocked due to enumeration attacks [9, 12].

One of the primary challenges associated with censorship evasion systems is distributing new Bridge "information" (e.g., IP addresses) to users who require it, especially those with limited access to uncensored regions and whose old Bridges might be blocked. This problem is commonly referred to as the Bridge Distribution Problem and has been extensively researched [8, 26, 30, 35]. We use an extended notion of "information" to encompass any data required for the establishment of a connection between the user and the Bridge, which may include more than just IP addresses, and divide the Bridge Distribution Problem into what we think are two adjacent subproblems: **I)** How can information on how to establish the covert connection be effectively transmitted through a communication channel between users and Bridges with a low risk of the communication channel being blocked, especially considering that the censor may possess the same knowledge of how to operate them as the user? **II)** How can we guarantee that the distributed Bridge information does not fall into the hands of censors or malicious users who could, in turn, block it?

This paper addresses the first subproblem described by designing a system based on a popular service available among Cloud providers, the Pub/Sub service, that can act as a rendezvous protocol [5, 32]. Rendezvous protocols or channels are censorship-resistant communication channels commonly used to bootstrap communication channels between users and Bridges. They leverage widely popular services or protocols that would cause significant collateral damage to most censors if blocked but are too costly (in terms of performance, price, or other metrics) to be used for regular traffic exchange. We implement a preliminary version of our system that facilitates indirect message exchange between the user and a Broker (i.e., a server holding information about available Bridges) by leveraging, specifically, the Google Pub/Sub service [20]. We then proceed to discuss the security of our system and present some preliminary results related to performance.

## 2 BACKGROUND

This section discusses our problem statement, how it relates to the Bridge Distribution Problem and the current state of rendezvous protocols.

### 2.1 Problem Statement

Most censorship evasion systems rely on users needing the IP address of Bridges to establish connections or on the exchange of

*Free and Open Communications on the Internet 2024 (2), 27–33*

information between the user and the Bridge to establish a connection [32]. Some systems assume users exchange this information through out-of-band channels, such as public and/or private forums, private chat rooms, and messaging applications [29]. Public forums are far from ideal because they can be easily identified and blocked, and the information they contain is publicly available. Although more restrictive than public forums, messaging apps, private forums, and private chat rooms require manual user effort and may only be a realistic option for some users. Moreover, in specific cases such as TorKameleon [31] or Snowflake [5], which use WebRTC connections to carry covert data, merely providing the IP address of the Bridge is insufficient to establish a censorship-resistant connection. Additional information (i.e., a signaling protocol) is required to establish the WebRTC connection between the user and the Bridge. The complexity of the bidirectional communication bootstrapping protocol in such systems makes it particularly challenging to use solutions that require manual effort. As such, the question of how to exchange or share such information is of great interest (i.e., the Bridge Distribution Problem [8, 26, 30, 35]). How can information about new Bridges, as well as other necessary information for establishing user-to-bridge connections, be shared and exchanged between users and Bridges in heavily censored regions in a manner that remains unblocked, even if the censor possesses the same information about the rendezvous protocol as the user?

In this paper, we are particularly interested in addressing the challenge of reaching users in “unreachable” regions due to censorship and providing them with the necessary information to connect with a Bridge. We consider this one of the two subproblems of the Bridge Distribution Problem. The second subproblem, preventing new Bridges from being blocked or ensuring that Bridges are only sent to trusted users, is left for future work. We address this first subproblem by developing a censorship-resistant rendezvous protocol that does not require secrecy to remain unblocked. Rendezvous channels or protocols [5, 32] leverage popular authorized services in the censored region as a front to carry covert information. They should, in theory, resist blocking attempts because blocking the services used by the rendezvous channels as carriers would result in collateral damage that the censor would be unwilling to accept.

### 2.2 Rendezvous Channels

Recent literature has showcased different mechanisms designed for rendezvous channels. One of the most popular methods is domain fronting [11]. The concept behind domain fronting is to enable a user to communicate with a blocked host through an intermediary such as a CDN. Domain fronting works by altering the externally visible hostname (SNI) from that of the censored content to a different “front domain”, making it appear that the user is connecting to an allowed host. The AMP framework, a framework for web pages written in a restricted dialect of HTML, has also been used as a rendezvous protocol. This framework includes a free-to-use cache server that can serve as an intermediary for a rendezvous channel, acting like a proxy. Similar to our system, Amazon’s messaging queuing service (Amazon SQS) is also used as a carrier for a rendezvous channel by exchanging messages through unidirectional queues [34]. Due to the nature of their covert channels, some censorship evasion systems can also function as rendezvous protocols. For example, CloudTransport [6] uses storage services from cloud providers to transmit information between a user and a Bridge. In [36], push notification services are used for censorship evasion. Email services are also used by systems such as the ones in [23, 33, 38]. MoneyMorph [24] uses cryptocurrencies to evade censorship. Finally, Raceboat [32] presents a framework that facilitates developing and managing rendezvous channels.

## 3 SYSTEM ARCHITECTURE

This section introduces the developed protocol, its implementation, the threat model, and the system’s security considerations.

### 3.1 Protocol Design

We present the architecture of the developed system in Figure 1. This system comprises four main actors: the user, the Broker, the Pub/Sub service, and the Bridge. The user represents the system user within the censored region. The Broker is a server that acts as a middleman between the Bridge and the user, holding information about Bridges and communicating directly with them on the user’s behalf. Having a Broker is advantageous as it allows for a more modular approach to rendezvous systems, rather than relying entirely on censorship evasion systems to accommodate the rendezvous protocol and implement its functionality [5]. The Pub/Sub service is the censorship-resistant communication channel and relays all messages between the user and the Broker. Finally, the Bridge works as the proxy for the censorship evasion system. When the user’s Bridge is blocked **(1)**, the user can initiate contact with the Broker using our protocol through the Pub/Sub service to acquire information about a new Bridge or exchange the information needed for a new Bridge connection **(2)**. The Pub/Sub service operates through topics and subscriptions. Subscribers subscribe to specific topics and only receive messages published on those topics. Once the user and the Bridge have the information needed, they can establish the new connection **(3)**. Our implementation uses the Google Pub/Sub service, but the design could, in theory, be modified and extended to be implemented using other services, and it currently only serves as a proof of concept. We explore different possibilities for cloud providers in Appendix C.

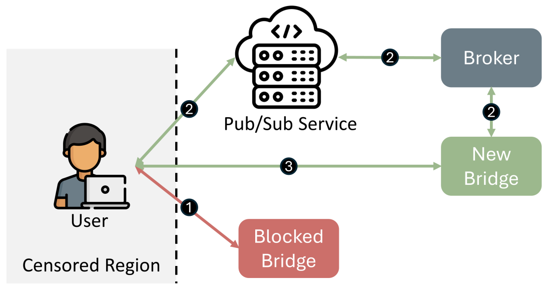

**Figure 1: System architecture.**

Our rendezvous protocol is outlined in Figure 2, and we follow the order of events depicted in the figure. Before delving into the protocol, it is necessary to establish some background information on our system. We use two predefined topics as unidirectional communication channels between users and Brokers. Users use one to send messages to the Broker (i.e., user's topic, **UT**), and Brokers use another to send messages to users (i.e., Broker's topic, **BT**). All users subscribe to the same topic and thus receive every message published by the Broker, whether intended for them or another user, while messages published by users will only be received by Brokers. We also assume that step **(0)** is done by each Broker once the Broker is deployed (during the first setup) and is needed because Brokers need to create their subscription to the **UT** topic so that they can receive messages published by users. For now, to simplify, we also assume only one Broker.

The initial step in the protocol is creating a subscription specific to the user for the **BT** topic, which is performed by the client-side software **(1)**. After creating the subscription, the user publishes a **CONNECT** message in the **UT** topic containing a randomly generated session ID (SID) **(2)**. The Broker receives it and publishes, in the **BT** topic, the message **PUBLIC_KEY** with its public key **(3)**. Upon receiving the message with the public key, the user creates a symmetric key and encrypts it with the Broker's public key, publishing a message **SYM_KEY** with the encrypted symmetric key and its SID in plaintext to the **UT** topic **(4)**. The Broker maintains the state of each session, specifically which step of the protocol the session is in and whether a Bridge and symmetric key have already been established for it. After performing step **(3)**, the Broker will wait for the symmetric key from the user who initiated the protocol, i.e., the user with the SID. Upon receiving the symmetric key and verifying that it is from the expected user by checking the SID, the Broker waits to obtain information about a new Bridge to send to the user **(5)**. This mechanism depends on the censorship evasion system and how Bridges communicate with the Broker. When the Broker has information to send or has found a Bridge for the user, it publishes the **HAS_BRIDGE** message along with the SID of the intended recipient **(6)** to the **BT** topic. The Broker and the user can now exchange the necessary information through the communication channels (i.e., the topics) by publishing **DATA** messages with their content (i.e., INFO) encrypted using the exchanged symmetric key and a symmetric encryption algorithm (e.g., AES-GCM). Finally, once all the information is exchanged, the user establishes the connection to the Bridge **(8)**. In Appendix A, we showcase an alternative design for this protocol.

## 3.2 Threat Model and Security Considerations

We trust the Broker and the cloud provider not to disclose any information they possess about the system at any given moment to censors. The Broker can associate any SID with its symmetric key and the chosen Bridge, provided that both have already been established (i.e., the Broker has access to the session's state). However, the Broker cannot infer which specific user (e.g., IP address) has a particular SID since SIDs are randomly generated by the client-side software each time the same or a different user initiates our protocol. Cloud providers can observe users' IPs and link them to their current SIDs, yet they cannot access their symmetric keys or identify the Bridge to which they are connected. We do not trust users and assume they may attempt to acquire information about other users, manipulate the protocol, obtain new Bridge information to subsequently block it, or employ DoS/DDoS attacks by flooding the system with requests, thereby throttling it or increasing operational

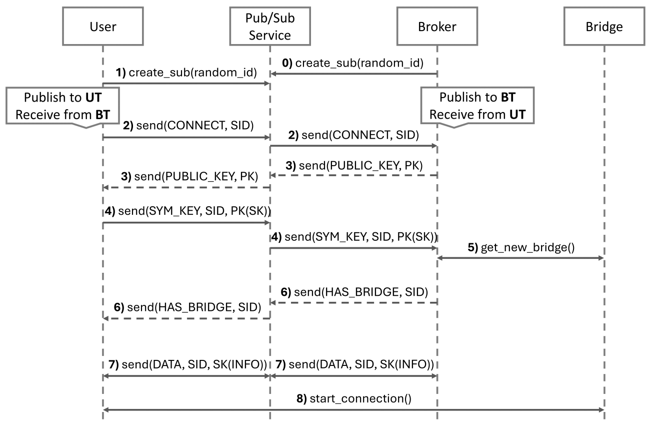

**Figure 2: Rendezvous Protocol design. The order of the exchanged messages is represented in numerical order.**

costs (we defer consideration of these last two types of attacks to future work).

Our protocol leaks the SIDs to all users listening on the **BT** topic in steps **(6)** and **(7)** (since all users share the same topic for receiving messages and can see all messages published by the Broker). However, we argue that this does not reveal anything about specific users, as users randomly generate their SIDs and have no actual link to the user. Users could still potentially impersonate other users by stealing SIDs. This would be possible if the SID was sent in plaintext to the **BT** topic before step **(4)**, which is not the case since the SID is only sent in plaintext to the **BT** topic for the first time in step **(6)**. However, at this point, the session already has a symmetric key established, and all subsequent messages published by the user of that session are encrypted with it, meaning that the impersonator would require the symmetric key to publish messages on behalf of the user. Our system also leaks when the Broker finds a Bridge for a session (in step **(6)**). However, this does not provide any information to malicious users since they do not know which Bridge is being used or who the user with a specific SID is. Nonetheless, this could be addressed by encrypting the message **HAS_BRIDGE**. Regarding access control, both our system's client-side and Broker software require access keys and credentials to interact with topics and subscriptions in a Google account via the Google Pub/Sub API. Thus, we use Google service accounts [14], enabling specific roles and permissions within the Google Cloud environment. A single Google service account is created for all users, with limited permissions. Users can only create subscriptions to the **BT** topic and publish messages to the **UT** topic, preventing malicious users from having more permissions and access than needed and ensuring that these credentials can be publicly available. Additional details on these accounts are provided in Appendix B.

## 3.3 Implementation

We have implemented a proof of concept of our system using the Go programming language. This prototype consists of two software components: the code for the Broker and the client-side software running locally on the user's device. The Pub/Sub service can be accessed through REST or gRPC requests to the service endpoints "https://pubsub.googleapis.com" or "pubsub.googleapis.com:443", respectively. In our implementation, we used the Go client library, opting for the gRPC library [17] instead of the REST API [16].

## 4 DISCUSSION

This section discusses the system's resilience to blocking and introduces some initial performance values.

### 4.1 Unblockability

Regarding the system's security, one of the primary objectives is to ensure that it remains unblocked for as long as possible. Blocking this system can essentially stem from three scenarios: **I)** The Google Cloud provider being blocked in the regions where the system is used; **II)** Instead of blocking all of the Google Cloud Platform's services, only the Pub/Sub service is blocked (e.g., by targeting the specific Pub/Sub service domain). **III)** By detecting a fingerprint in the traffic generated by our system (we leave this for future work).

We view "unblockability" as a function that takes as input a system, service, or protocol and generates an output value. This output represents the cost to the censor of blocking that particular system, service, or protocol, and it heavily depends on the censor's motivations and what they are willing to sacrifice [5]. We also assume that the censor defines a third value as the threshold of collateral damage and population discontent that he is unwilling to exceed. If the output value of the function exceeds this threshold, then the system, service, or protocol cannot be blocked. The problem is that different censors can produce varying outputs even for identical inputs, making it challenging to generalize and estimate a system's overall unblockability. Thus, we must rely on a heuristic approach, examining censors' past actions and current censorship practices and observing which systems remain unblocked for extended periods and which do not. Those that have not been blocked or are widely popular in censored regions are ideal candidates to be used in communication channels to evade censorship. The most used cloud providers, such as Amazon AWS, Microsoft Azure, and Google Cloud, are strong candidates due to their widespread popularity and the significant reliance that businesses and organizations place on their services. However, instances of their blockage exist globally. For example, Google Cloud services are inaccessible in China. Yet, we argue that developing rendezvous protocols that might be restricted in specific authoritarian regimes still holds some value as long as they work in others. Having rendezvous protocols that only work in particular regimes could benefit populations within such regimes, ensuring a broader safety net of censorship-resistant services. Additionally, we argue that Pub/Sub services are strong candidates as carrier protocols due to their significant role in communication models between IoT devices, Edge, and Cloud servers. The growing trend of IoT devices across various contexts (e.g., smart homes, retail, and industries) further strengthens the popularity of Pub/Sub systems. We also provide examples of applications that use, specifically, the Google Pub/Sub service in Appendix B.

### 4.2 Performance

We present preliminary results regarding the time it takes for a censorship evasion system, i.e., TorKameleon, to establish a connection to a Bridge using our system. Firstly, we calculated the time it takes to connect to a TorKameleon Bridge without our system. In this scenario, a special server, also known as the Broker, facilitates the relay of information between the Bridge and the user. They must communicate to exchange network information, media capabilities, and other configurations necessary to establish a peer-to-peer WebRTC connection. This value serves as the baseline for our evaluation. Subsequently, we conducted the same test, but this time, we used our system as an intermediary to exchange the information needed for the connection.

| | A (s) | B (s) | Total Time (s) |
|---|---|---|---|
| Baseline | 1.32 | - | 1.32 |
| Our System | 3.26 | 5.23 | 8.49 |

**Table 1: Bootstrapping time comparison (seconds). A: Time for message exchange. B: Time for subscription creation.**

The test bench consisted of two machines. The first one was located in Europe, connected to the network through a 1Gbps bandwidth connection, and ran the client-side software. It had an Intel Core i5 9300H processor with 4 cores (2.40 GHz) and 16 GB of RAM. The Broker (both the one used in the baseline and the one from our system) and the TorKameleon Bridge were run on a virtual private server (VPS) in the East US region, with 1 core and 1 GiB of RAM. We did not find the bandwidth values for the VPS used. Both systems had Ubuntu 20.04. Between the machine in Europe and the one in the East US, there was an average latency of 113 ms. The results can be observed in Table 1. Each measurement presented is the average of 10 samples. For the tests regarding our system, we divided the time into two subcategories which together make up the total time of bootstrapping TorKameleon: the subscription creation time (step **(1)** in Figure 2) and the time it takes to exchange the information needed for the connection (steps **(2)**-**(8)** in Figure 2). As shown by the results, the primary bottleneck of the system in terms of time consumption is the subscription creation part. This delay is primarily attributed to the time required for Google Cloud to create the new subscription resource. Additionally, the message exchange time for our system is approximately 2 seconds longer than the baseline, which remains within reasonable bounds.

## 5 CONCLUSION

This article proposes a solution for users in censored regions with limited internet access to obtain the necessary information to establish a connection with a Bridge. Our system uses Pub/Sub services as data carriers, enabling real-time indirect message exchange between users and a Broker, which acts as an intermediary to Bridges.

Our future work aims to extend our design to integrate various Pub/Sub services across different cloud providers. This task involves adapting our protocol to accommodate other Pub/Sub services and integrating new functionalities, including broker and user authentication. Additionally, we intend to evaluate the system for traffic analysis to detect patterns (fingerprints) and assess performance metrics, such as scalability.

## ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their constructive feedback. The authors would also like to acknowledge the support provided by FCT through the PhD grant (PRT/BD/154787/2023) and by NOVA LINCS (UIDB/04516/2020 and UIDP/04516/2020) with the financial support of FCT, I.P.

## REFERENCES

[1] Yawning Angel. 2014. *obfs4*. https://gitlab.com/yawning/obfs4

[2] AWS. -. *Amazon Web Services (AWS) - What Is Pub/Sub Messaging?* https://aws.amazon.com/what-is/pub-sub-messaging/?nc1=h_ls

[3] Diogo Barradas, Nuno Santos, Luís Rodrigues, and Vítor Nunes. 2020. Poking a Hole in the Wall: Efficient Censorship-Resistant Internet Communications by Parasitizing on WebRTC. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 35–48. https://doi.org/10.1145/3372297.3417874

[4] Zachary S. Bischof, Kennedy Pitcher, Esteban Carisimo, Amanda Meng, Rafael Bezerra Nunes, Ramakrishna Padmanabhan, Margaret E. Roberts, Alex C. Snoeren, and Alberto Dainotti. 2023. Destination Unreachable: Characterizing Internet Outages and Shutdowns. In *Proceedings of the ACM SIGCOMM 2023 Conference* (New York, NY, USA) *(ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 608–621. https://doi.org/10.1145/3603269.3604883

[5] Cecylia Bocovich, Arlo Breault, Xiaokang Wang, David Fifield, and Serene. 2024. Snowflake, a censorship circumvention system using temporary WebRTC proxies. In *33nd USENIX Security Symposium (USENIX Security 24)*. USENIX Association. https://www.bamsoftware.com/papers/snowflake/snowflake.pdf

[6] Chad Brubaker, Amir Houmansadr, and Vitaly Shmatikov. 2014. CloudTransport: Using Cloud Storage for Censorship-Resistant Networking. In *Privacy Enhancing Technologies*, Emiliano De Cristofaro and Steven J. Murdoch (Eds.). Springer International Publishing, Cham, 1–20.

[7] Roger Dingledine and Nick Mathewson. 2006. *Design of a blocking-resistant anonymity system.* Technical Report. The Tor Project. https://svn.torproject.org/svn/projects/design-paper/blocking.pdf

[8] 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://petsymposium.org/popets/2016/popets-2016-0026.pdf

[9] Roya Ensafi, David Fifield, Philipp Winter, Nick Feamster, Nicholas Weaver, and Vern Paxson. 2015. Examining How the Great Firewall Discovers Hidden Circumvention Servers. In *Proceedings of the 2015 Internet Measurement Conference* (Tokyo, Japan) *(IMC '15)*. Association for Computing Machinery, New York, NY, USA, 445–458. https://doi.org/10.1145/2815675.2815690

[10] 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 *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 2599–2616. https://www.usenix.org/conference/usenixsecurity23/presentation/feng

[11] 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

[12] David Fifield and Lynn Tsai. 2016. Censors' Delay in Blocking Circumvention Proxies. In *6th USENIX Workshop on Free and Open Communications on the Internet (FOCI 16)*. USENIX Association, Austin, TX. https://www.usenix.org/conference/foci16/workshop-program/presentation/fifield

[13] Gabriel Figueira, Diogo Barradas, and Nuno Santos. 2022. Stegozoa: Enhancing WebRTC Covert Channels with Video Steganography for Internet Censorship Circumvention. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security* (Nagasaki, Japan) *(ASIA CCS '22)*. Association for Computing Machinery, New York, NY, USA, 1154–1167. https://doi.org/10.1145/3488932.3517419

[14] Google Cloud. -. *Google Cloud IAM Service Account Overview Documentation.* https://cloud.google.com/iam/docs/service-account-overview

[15] Google Cloud. -. *Google Cloud Pub/Sub Pricing Documentation.* https://cloud.google.com/pubsub/pricing#pubsub

[16] Google Cloud. -. *Google Cloud Pub/Sub REST API Reference Documentation.* https://cloud.google.com/pubsub/docs/reference/rest

[17] Google Cloud. -. *Google Cloud Pub/Sub RPC Reference Documentation.* https://cloud.google.com/pubsub/docs/reference/rpc

[18] Google Cloud. -. *Google Cloud Pub/Sub Service APIs Overview Documentation.* https://cloud.google.com/pubsub/docs/reference/service_apis_overview

[19] Google Cloud. -. *Google Cloud Resource Manager Documentation: Creating and Managing Projects.* https://cloud.google.com/resource-manager/docs/creating-managing-projects

[20] Google Cloud. 2024. Google Cloud Pub/Sub. https://cloud.google.com/pubsub?hl=en.

[21] Sergei Guriev and Daniel Treisman. 2020. A theory of informational autocracy. *Journal of Public Economics* 186 (2020), 104158. https://doi.org/10.1016/j.jpubeco.2020.104158

[22] Haifeng Huang and Nicholas Cruz. 2022. Propaganda, Presumed Influence, and Collective Protest. *Political Behavior* 44, 4 (01 Dec 2022), 1789–1812. https://doi.org/10.1007/s11109-021-09683-0

[23] Shuai Li and Nicholas Hopper. 2016. Mailet: Instant Social Networking under Censorship. *Privacy Enhancing Technologies* 2016, 2 (2016), 1–18. https://petsymposium.org/popets/2016/popets-2016-0011.pdf

[24] Mohsen Minaei, Pedro Moreno-Sanchez, and Aniket Kate. 2020. MoneyMorph: Censorship Resistant Rendezvous using Permissionless Cryptocurrencies. *Privacy Enhancing Technologies* 2020, 3 (2020), 404–424. https://petsymposium.org/2020/files/papers/issue3/popets-2020-0058.pdf

[25] Hooman Mohajeri Moghaddam, Baiyu Li, Mohammad Derakhshani, and Ian Goldberg. 2012. SkypeMorph: protocol obfuscation for Tor bridges. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (Raleigh, North Carolina, USA) *(CCS '12)*. Association for Computing Machinery, New York, NY, USA, 97–108. https://doi.org/10.1145/2382196.2382210

[26] Milad Nasr, Sadegh Farhang, Amir Houmansadr, and Jens Grossklags. 2019. Enemy At the Gateways: Censorship-Resilient Proxy Distribution Using Game Theory. In *Network and Distributed System Security*. The Internet Society. https://people.cs.umass.edu/~amir/papers/TorGame.pdf

[27] Sadia Nourin, Van Tran, Xi Jiang, Kevin Bock, Nick Feamster, Nguyen Phong Hoang, and Dave Levin. 2023. Measuring and Evading Turkmenistan's Internet Censorship: A Case Study in Large-Scale Measurements of a Low-Penetration Country. In *Proceedings of the ACM Web Conference 2023* (Austin, TX, USA) *(WWW '23)*. Association for Computing Machinery, New York, NY, USA, 1969–1979. https://doi.org/10.1145/3543507.3583189

[28] 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 *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 2581–2598. https://www.usenix.org/conference/usenixsecurity23/presentation/ramesh-network-responses

[29] The Tor Project. -. *Tor Bridges.* https://bridges.torproject.org/

[30] Lindsey Tulloch and Ian Goldberg. 2023. Lox: Protecting the Social Graph in Bridge Distribution. *Privacy Enhancing Technologies* 2023, 1 (2023). https://petsymposium.org/popets/2023/popets-2023-0029.pdf

[31] Afonso Vilalonga, João S. Resende, and Henrique Domingos. 2023. TorKameleon: Improving Tor's Censorship Resistance with K-anonymization and Media-based Covert Channels. arXiv:2303.17544 [cs.CR]

[32] Paul Vines, Samuel McKay, Jesse Jenter, and Suresh Krishnaswamy. 2024. Communication Breakdown: Modularizing Application Tunneling for Signaling Around Censorship. *Privacy Enhancing Technologies* 2024, 1 (2024). https://www.petsymposium.org/popets/2024/popets-2024-0027.pdf

[33] Ryan Wails, Andrew Stange, Eliana Troper, Aylin Caliskan, Roger Dingledine, Rob Jansen, and Micah Sherr. 2022. Learning to Behave: Improving Covert Channel Security with Behavior-Based Designs. *Proceedings on Privacy Enhancing Technologies* 2022 (07 2022), 179–199. https://doi.org/10.56553/popets-2022-0068

[34] Andrew Wang, Anthony Chang, Kieran Quan, Michael Pu, Yi Wei Zhou, and Cecylia Bocovich. 2024. New SQS rendezvous method for Snowflake. https://github.com/net4people/bbs/issues/335 Accessed: 04 12, 2024.

[35] Qiyan Wang, Zi Lin, Nikita Borisov, and Nicholas J. Hopper. 2013. rBridge: User Reputation based Tor Bridge Distribution with Privacy Preservation. In *Network and Distributed System Security*. The Internet Society. https://www-users.cs.umn.edu/~hopper/rbridge_ndss13.pdf

[36] Diwen Xue and Roya Ensafi. 2023. The Use of Push Notification in Censorship Circumvention. In *Free and Open Communications on the Internet*. https://www.petsymposium.org/foci/2023/foci-2023-0009.pdf

[37] Tarun Kumar Yadav, Akshat Sinha, Devashish Gosain, Piyush Kumar Sharma, and Sambuddho Chakravarty. 2018. Where The Light Gets In: Analyzing Web Censorship Mechanisms in India. In *Internet Measurement Conference*. ACM. http://delivery.acm.org/10.1145/3280000/3278555/p252-Yadav.pdf

[38] Wenxuan Zhou, Amir Houmansadr, Matthew Caesar, and Nikita Borisov. 2013. SWEET: Serving the Web by Exploiting Email Tunnels. In *Hot Topics in Privacy Enhancing Technologies*. Springer. https://petsymposium.org/2013/papers/zhou-censorship.pdf

## A PROTOCOL AND SYSTEM DESIGN

### A.1 Alternative Protocol Designs

In this section, we introduce an alternative to the designed protocol, depicted in Figure 3. This alternative bears similarities to the original protocol but introduces some key differences. In this version, instead of relying on a global topic for all users, the Broker creates specific topics and subscriptions that act as private communication channels for each user. Therefore, the user is subscribed to their specific topic, ensuring that only that specific user receives messages intended for them. Nevertheless, it's worth noting that this version has not undergone complete testing or implementation.

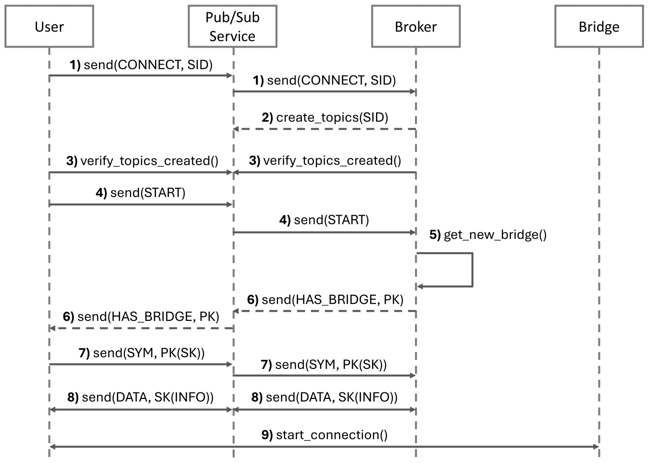

**Figure 3: Alternative protocol design for the rendezvous channel. The order of the exchanged messages is represented in numerical order.**

Both messages **2** and **3** are new compared to the regular version of the protocol. Upon receiving the initial request to bootstrap the system from the user, the Broker creates two new topics with names based on the received SID **(2)**: one for the Broker to send messages to the user and another for the user to send messages to the Broker, both acting as unidirectional private communication channels. It also establishes two subscriptions, one for each topic, allowing the user and the Broker to read from their respective channels. Using two topics instead of just one per client ensures that both the Broker and the user do not receive the messages they send to each other, avoiding the scenario where both parties act as publishers and subscribers on the same topic used as the private communication channel. Once the Broker and the user verify that the topics and their subscriptions have been created **(3)**, the message exchange can begin. The user sends a **START** message to the Broker **(4)**, and the rest of the protocol follows a similar pattern to the regular version. In this version, the primary source of time consumption occurs when the topics and their respective subscriptions are created for each user's connection, and their creation is verified (steps **(2)** and **(3)** in Figure 3). In our testbench, the average time taken for each topic and its respective subscription to be created is **10.81** seconds.

## B GOOGLE PUB/SUB SERVICE

### B.1 Service Accounts

Service accounts [14] are a special type of account typically used by applications rather than by a person to access Google Cloud services and make authorized API calls. Service accounts are principals, meaning granting them access, permission, and roles in Google Cloud resources is possible. Service accounts can be defined within projects. Projects [19] are a way to organize resources within a Google Cloud account and allow the management of permissions for the Google Cloud resources existing within a project.

We create two projects: one for the **BT** topic and its associated subscriptions (referred to as Project One) and another for the **UT** topic and its associated subscriptions (referred to as Project Two). Then, we create a service account for each project: Service Account A for Project One and Service Account B for Project Two. Service account A will have permission to create subscriptions to the **BT** topic and consume messages using those subscriptions. Additionally, service account A will have permission to publish messages on the topic of Project Two, the **UT** topic. This service account serves as the user account. On the other hand, service account B will have permission to create subscriptions to the **UT** topic and consume messages using those subscriptions. Additionally, service account B will have permission to publish messages on the topic of Project One, the **BT** topic. This service account serves as the Broker account. Both accounts only have permissions for what they require. We use two projects because having only one project for both accounts and all resources would permit users to create subscriptions in the project where the **UT** topic is defined, potentially allowing them to read messages intended for Brokers.

### B.2 Credentials

Below, in Listing 1, is an example template of the credentials JSON file required for the system to interact with the Pub/Sub service.

```
{
  "type": "service_account",
  "project_id": "project_id",
  "private_key_id": "PRIVATE_KEY_ID",
  "private_key": "PRIVATE_KEY",
  "client_email":
      "project_id@appspot.gserviceaccount.com",
  "client_id": "ID",
  "auth_uri":
      "https://accounts.google.com/o/oauth2/auth",
  "token_uri":
      "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url":
      "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url":
      "googleapis.com/robot/v1/metadata/x509/
client_email",
  "universe_domain": "googleapis.com"
}
```

**Listing 1: JSON Credentials File**

To use our system, users need this information, as well as the name of the **UT** topic, the **BT** topic and the project ID associated with the **UT** topic.

### B.3 Regions

Requests sent to the global endpoint of the Pub/Sub service from outside the Google Cloud are routed to a nearby available region. However, users can also send requests to one of the available Pub-/Sub service regions using the locational Pub/Sub endpoints [18]. Storage policies can also be applied to ensure that the processing and storage of messages occur only in the regions specified in this policy, regardless of the origin of the publish requests. If no storage policy or resource location restriction organization policy is defined, messages can be stored and processed in any available Google Pub/Sub region.

### B.4 Pricing

The Google Pub/Sub service requires a Google Cloud account. The account and the Pub/Sub service can be created and utilized for free, although the Pub/Sub service is only free for message volumes of up to 10GB per month [15].

### B.5 Usecases

We present two examples of real-world applications that use the Google Cloud Platform Pub/Sub service (both examples were taken from the Google Cloud Pub/Sub website and showcased by Google themselves [20]). The first example was developed by the CME Group, which used Google Pub/Sub to provide customers with a real-time messaging market data service. This service allows consumers to subscribe to topics where messages and events related to market data are published. Sky, a media and communications company, implemented the second example, using Google Pub/Sub to transmit diagnostic data from TV boxes (IoT devices) to their infrastructure. Both examples illustrate the application of the Pub/Sub service in consumer-oriented contexts.

## C OTHER PUB/SUB SERVICES

We leveraged the Google Pub/Sub service for our prototype. Yet, other cloud providers offer similar services that we intend to explore in the future, such as Amazon Simple Notification Service (SNS) [2]. Amazon SNS only permits direct upstream connections between user devices and Amazon SNS. In other words, any device can publish messages, but only specific endpoints can subscribe to topics and receive published messages. Specifically, SNS supports delivering messages to other Amazon services (e.g., Lambda functions and Amazon SQS queues), HTTP(S) endpoints, mobile text messaging, or email. Adapting our system to Amazon SNS will require implementing a different method for downstream delivery to users. Microsoft Azure also offers Pub/Sub services. However, unlike Amazon and Google Cloud Pub/Sub services, Azure's services have specific domains assigned to each created resource. This characteristic makes them easier targets for censors to block.