Evaluation of Proxy Distribution Mechanisms in the Wild

Syed Mahbub Hafiz
Indiana University–Bloomington
shafiz@indiana.edu

Sadia Afroz
ICSI
sadia@icsi.berkeley.edu

Damon McCoy
New York University
mccoy@nyu.edu

I. Introduction

The anonymous communication system Tor is used as a tool to circumvent censorship. A vital challenge to employ this tool is to distribute the Tor bridges to censored users. There are several bridge distribution mechanisms, but all are implemented and evaluated in a simulated scenario. We plan to assess the state-of-the-art proxy (Tor Bridge) distribution mechanisms and inspect the performances in the real Tor.

In this proposal, we discuss the proxy distribution mechanisms in Section II and tabulates the evaluation metrics to compare them with each other in Section III. We describe the capacity of real-world censors and the reachability of Tor bridges in Section V and IV, respectively. Next, we inspect the system parameters of the mechanisms keeping an eye to find the minimum settings to run them in Section VI. We conclude by outlining how we can redesign the current Tor bridge distributor, according to rBridge [9] and Hyphae [5], to conduct our evaluation plan.

II. Related work

We reviewed 4 essential papers on this topic.

A. Proximax [6]:

Proximax distributes pools of proxies widely to many channels (eventually end-users), which maximizes the usage-hours (how many end-users a proxy serves and the lifetime of it) of the proxies and minimizes the risk of getting blocked by the censors. To achieve the first task, it delegates the advertisements (via different channels like the private mailing list, social networking, or in person) of the proxies to the most useful set of registered users, where the effectiveness of a registered user is evaluated by an analytical method.

In the ecosystem, there is a trusted group of administrators who have the list of all proxies who offers individualized hostname, i.e., a unique domain name registered with DNS, to the registered user that supports to estimate how many end-users the registered user (channel) brings to a set of proxies. The channel (that has a domain name) consists of a growing collection of proxies that gets a new hostname once the incumbent hostname gets compromised.

An already registered user can invite a new user by evaluating a reputation-based analysis of the subtree of the inviting registered user (being the root of the subtree). The performance of the root and each subnode produces a reputation score for the root based on the user-hours and number of blocked resources throughout the subtree, where a lower score denies the new invitation and vice-versa.

In the analytical model, each channel imposes some risk to the proxies, quantified by the Poisson Process. They consider the channels independent of each other, which keeps the combined risk, due to distribute one proxy over multiple channels, as the additive in operation.

B. rBridge [9]:

rBridge has a user reputation system (based on the user) for Tor bridge distribution, preserving the privacy of the user's bridge assignment information and proposing an invitation-based method to include new users while resisting Sybil Attacks. Censors can run an insider attack by colluding with corrupt users and can amplify it further by deploying Sybils to enumerate the bridges. When only a subset of the bridges is assigned to a user, an adversary can fingerprint that user after monitoring the bridges, so privacy-preserving bridge distribution (as well as reputation system) is required. Bridge distributor doesn't know which bridges are given to which user, which enables the user to be malicious controls its reputation score.

The user reputation system of rBridge provides a reputation score on the uptime of its assigned bridges. If one of the designated bridges got blocked, the user could replace it by paying a credit from its score. rBridge targets to maximize user-hours, minimize thirsty-hours, keep user base growth healthy, and preserve the bridge assignment's privacy. In the threat model, a registered malicious user (insider attacker) can handover the information of the allotted bridges to a censor to make them blocked. A bridge distributor is an honest but curious entity. There is a trade-off between the openness of the system and robustness to the insider attacks.

C. TorBrix [10]:

TorBricks describes a resource-competitive analysis based bridge distribution mechanism that makes Tor available to all honest users without prior knowledge of the number of corrupt users. It guarantees that O($t \log n$) number of bridges need to distribute to reach all honest users after maximum O($\log t$) number of rounds, where n is the total number of, and t is the number of corrupt users. It can work with Tor without any modification to it, and it offers privacy-preserving bridge distribution.

The number of distributed bridges increases the number of blocked bridges. Users can join and leave the system often.

The bridges are allocated to the users by pseudonyms, which preserve the privacy of bridge assignment information against honest-but-curious distributors. It employs a Distributed Random Generation protocol to disseminate the bridges among the untrusted distributors using a secret sharing scheme, which can withhold collusion up to one-third of the distributors. The protocol employs bridge reachability to test blocked bridge from outside the censored regime. Tor develops pluggable transcripts to obfuscate the traffic communicated between users and bridges, which prevents the Deep Packet Inspection attack.

In their evaluation, they use several performance metrics, such as the number of thirsty users, the number of distributed bridges, the number of blocked bridges in the current round, and the total number of unique bridges distributed by the protocol until this round. The number of rounds required until all honest users get at least one bridge.

D. Enemy At the Gateways: [7]

This manuscript considers optimal-censors with advanced censorship strategies. It evaluates the performance by considering various adversarial and network settings. It models the proxy assignment problem as a stable matching problem (college admission game) using the deferred acceptance approach between colleges (proxies) and students (censored clients including censoring agents) where the entities rank each other using representing metrics. The game theory identifies the optimal proxy distribution strategy for a circumvention system and the optimal censorship strategy for the censors. In the following, we highlight the critical system parameters and considerations they make.

- System parameters:
  - System's lifespan covers The birth interval (the initial phase until it reaches a stable rate of growth) and the stable interval (right after the birth interval).
  - ecosystem considers the rate of new clients per time unit and The rate of new proxies per time unit during the birth interval and after the birth interval.
  - The distribution of censoring agents: they consider two different types of censors.
    - The omnipresent censor, a resourceful entity, can run censoring agents at various geographic locations.
    - The circumscribed censor runs its censoring agents inside a single subnet.
  - Users' locations: censorship region is rectangular. Both kinds of users (censored and censoring) are uniformly distributed over this region.
  - Proxy parameters: Each user gets three proxies. The capacity of each proxy is 40 users.
  - Scaling factors for the utility functions.
    - It let benign users increase their utility by using their proxies over long periods. It prevents censoring agents from increasing their utility.
    - It allows censored clients to request new proxies.
    - It considers asking for a new proxy while having an available proxy is suspicious to the distributor.
    - It punishes users who have many of their assigned proxies blocked.
    - It assumes that blocking users is more important to the censors than keeping the censoring agents alive,
    - It incurs a high cost for losing agents,

III. Evaluation criteria and metrics

Table I presents the adversaries each method considers in their work, and Table II depicts the evaluation parameters variance concerning user metrics and bridge or proxy metrics over the processes. The last column, "Latency," shows how long a system takes to give at least an unblocked proxy to all honest users. Torbrix [10] presents this metric in the required number of rounds against the number of corrupt users, and Enemy At the Gateways [7] demonstrates the number of rounds necessary to avail the same condition against the number of days. So for the lack of availability of data of the same setup, instead of mentioning a number, we cite whether the protocol considers it or not.

IV. Tor bridge reachability methods

Bridge distributor needs to check whether a bridge has been blocked in a given region. There are particular Tor bridge reachability testing methodologies in the literature. In this section, we revised 2 recent papers from Free and Open Communications on the Internet (FOCI).

A. 2016 Status [2]

The authors measured the bridge's reachability from some sites in the U.S., China, and Iran before and after the first public disclosure of the bridges in Tor Browser release. A censored user can use a pluggable transport to contact a secret bridge by circumventing deep packet inspection and IP address blacklisting. Though the default bridges are open to the world in the Tor Browser source code, not many censors (unlike China's Great Firewall) block them immediately. That's why Tor Browser continues to publish default bridges.

The process of releasing a new bridge includes ticket filing, ticket merging, release testing, and finally, the public release. From ticket filing to public release, the process usually takes a few weeks. At each site of China and Iran, they ran TCP reachability tests to various destinations every 20 minutes. For each target, the script tried to establish a TCP connection and then record the current time, IP address, port number, status, and error message. They measured four bridges that appeared only in Orbot, the port of Tor to Android. The Orbot-only bridges remained accessible, even as the Tor Browser bridges were blocked.

The results from their two China sites are the same in most respects, while blocking happened concurrently. They found no blocking at all of the default bridges from the site in Iran. Blocking of the new bridges gets delayed, but abrupt when a batch contained more than one bridge. Across the five batches in the Tor Browser releases, they observed blocking delays of $7, 2, 18, 11$, and 36 days after the first public release,

Table I. THREAT MODEL

Methods	Adversaries/Censors				
	Aggressive	Conservative			Optimal
		Prudent	Stochastic	Event-driven	
rBridge [9]	✓	✓	✓	✓	✗
TorBricks [10]	✓	✓	✓	✗	✗
Enemy At the Gateways [7]	✓	✓	✓	✗	✓

Table II. EVALUATION METRICS

Methods	Evaluation metrics						
	Users				Bridges		Latency
	User-hours	Number	Growth	Thirsty-users	Distributed	Blocked	
rBridge [9]	✓	✗	✓	✓	✗	✗	✗
TorBricks [10]	✗	✗	✗	✓	✓	✓	✓
Enemy At the Gateways [7]	✗	✓	✗	✓	✓	✗	✓

and up to 57 days after the filing of the first ticket, when bridges were potentially first discoverable. This fact suggests that new default bridges are loaded into the firewall in groups, and are not detected and blocked serially. From these facts, they inferred that the censors in China learn of bridges not from the bug tracker, nor the source code inspection, but only from public releases. The variable delay in blocking over batches suggests the blocking procedure is partly manual as well as automatic. The manual process discovers bridges after a *unpredictable* delay; then, a periodic, automatic process causes the blocks of them to take effect. Therefore, they recommended that if the censor blocks new bridges after n days, the circumvention system can introduce new bridges every $n - 1$ days.

B. 2018 status [1]

Authors investigated the Tor network's reachability from censored regions, how unpublished bridges got blocked, how Great Firewall scanners have upgraded recently, and what could be an effective circumventions strategy for a censored client. GFW China combines deep packet inspection (DPI) with active scanning to identify unpublished bridges. When a client connects to such a bridge, the GFW uses DPI to check if the link is a Tor traffic. If yes, the firewall assigns multiple scanners to test whether the relay is running the Tor service. The authors employed vantage points of three Tor bridge relays and one Tor client in China. They used the Tor Connection Initiation Simulator (TCIS) to test the Tor network's reachability and their Tor bridges from the client in China. The reachability tests comprise the attempt to retrieve the Tor consensus, the effort to connect to the published relays, monitoring the blocking of published relays, the attempt to communicate to their bridge relays, tracking for GFW scanners, and pinging relays and bridges.

When a Chinese client tried to connect public tor relays, the connection attempt got blocked by receiving a TCP reset packet from GFW. When they published their relays in Tor consensus and continued pinging from China, after 10 minutes of publication, the IP addresses of the relays were added to an IP blacklist. When their relays stopped Tor service, the unblocking didn't happen immediately; instead, it took 12 hours to get free from the blacklist.

The Chinese client ran TCIS to connect the unpublished bridges from China. Like published relays, the attempt received a TCP reset packet, and after some time, the bridge got blocked. Other web services at different ports became inaccessible from China, which shows the firewall blocked the IP address entirely regardless and costs co-lateral damage in cloud services. The IPs collected from the Tor consensus are kept into the same blacklist as bridge relays. They also found that if they ran TCIS during the block to any of the blocked bridges, the block duration renews to 12 hours from then. These observations prove the firewall tries to save resources.

Once a connection attempt is made by the client to the relay, and a scanner is active, the relay's IP address isn't scanned again for next 12 hours. To exploit this incidence, their bridge relay can drop packets from scanners so that the scanner does not confirm that the relay is a bridge relay and does not add the relay to the blacklist. This allows them to repeatedly initiate Tor connections to the relay from the client and observe scan attempts, which shows the network is a distribution of proxies that forward packets from a centrally controlled system.

meek based on domain fronting routes traffic through a CDN before the traffic is sent to a bridge. They tested *meek* with the built-in public Azure server and successfully established a Tor circuit. Using *obfs4*, i.e., based on obfuscation rather than domain fronting, they were able to create Tor circuits on their bridges, but are unable to connect to public bridges. This is likely due to public bridges being recorded and blacklisted by the GFW.

They found that the easiest method for users to bypass the GFW is *meek*. Additionally, *obfs4* is easier to deploy on an unpublished bridge than *meek*, as certificate management is not required. However, if the bridge distribution can be performed in a controlled way, rejecting scanners is equally viable, significantly more cost-effective, and even bridge operators can implement the rejection feature natively.

V. REAL-LIFE CENSORSHIP

After the discussion of Section IV, now we summarize the capacity of Censors in the wild. Real-life censors can employ three critical kinds of adversaries based on network-level attack, insider attack, and Sybil attack to discover bridges. The state-of-the-art proxy or bridge distribution mechanisms [6, 9, 10, 7] can circumvent the insider attack (as well as Sybil attacks) in different capacities. However, they cannot bypass the censorship related to network-level bridge adversaries, such as Deep Packet Inspection and Active Probing attacks. To

justify this failure, we recite the following statements from the papers.

Wang et al, [9] said,

> For clarity, we do not attempt to address network-level bridge discovery. We assume the censor can learn bridges only from the distribution channels (i.e., based on the knowledge of registered corrupt users and Sybils). The censor may employ other techniques to discover bridges. For instance, the censor could try to probe all IP addresses on the Internet to find hosts that run Tor handshake protocols, fingerprint Tor traffic to identify bridges or monitor the users who connect to a discovered bridge to see what other TLS connections these users establish and try to verify further whether the connected hosts are bridges. If the censor were able to identify bridges using such network-level bridge discovery techniques, any bridge distribution strategy would not be able to work.

Zamani et al. mentioned in [10],

> The Tor Project has developed a variety of tools known as pluggable transports to obfuscate the traffic transmitted between clients and bridges. This makes it hard for the censor to perform deep packet inspection (DPI) attacks since distinguishing actual Tor traffic from legitimate-looking obfuscated traffic is hard. The censor can also block bridges using active probing: he can passively monitor the network for suspicious traffic, and then actively probe dubious servers to block those determined to run the Tor protocol. We believe that active probing will be defeated using a combination of ideas from CAPTCHAs, port knocking, and format transforming encryption. Depending on the sophistication of the censor, TorBricks may be used parallel with tools that can handle DPI and active probing to provide further protection against blocking.

and Nasr et al. discussed in [7],

> We assume that the clients have no way of obtaining the proxy information other than contacting the distributor entity, e.g., a censored client will not be able to obtain a proxy IP by asking friends. Similarly, we assume that the censoring agents can only obtain proxy information by requesting the circumvention distributor. Therefore, we do not consider other adversarial means of discovering proxies: probing suspicious IPs, zig-zag attacks, and other forms of active attacks in this work. Future work can trivially include any of these techniques into our generic game-theoretic model (i.e., by modifying our utility functions described in the following sections); we refrain from doing so in this work to avoid over-complicating the results.

According to the study of [1] regarding the reachability tests of Tor relays, the real-life censor (like the Chinese Great Firewall) runs DPI and Active Probing Attacks. As soon as any client from a censored regime attempts to connect any unpublished Tor Bridge, i.e., received from Bridge Distributor by the underlying distribution method, the client receives a TCP reset packet. Eventually, the bridge starts receiving multiple probing attacks from the censoring scanners while the censor wants to check whether the server supports Tor service.

There are several ways to evade these network-level attacks from the real-time censors. Pluggable transport, e.g., *meek, obfs4,* etc., is one of them which turns Tor traffic from censored clients into indistinguishable one. Another possible suggestion from [1] is that the bridge can drop the packets from GFW so that scanners don't make sure the bridge is servicing Tor. Therefore, if we want to circumvent real-life Censors, we have to enroll one of these techniques to mitigate network-level attacks at first. In the next phase, we can employ a sophisticated Bridge distribution strategy to evade an insider attack or Sybil attack. That is one reason why do the methods [6, 9, 10, 7] keep network-level adversaries out of scope.

VI. Minimum number of parameter values

We discuss a tentative plan to run the evaluation of each of the methods with minimum values of the system parameters.

A. rBridge

a) Protocol: $g = 40$ is the number of users a bridge can service. The system starts with N users. f the fraction of malicious users. Each user receives $k = 3$ bridges. T_0 and T_1 denote the expected lower and upper bounds of a bridge's life time. s presents the rate of recruiting new bridges. $T_0 = \frac{N \cdot k}{g \cdot s_{max}}$ and $T_1 = \frac{N \cdot k}{g \cdot s_{min}}$. $s_{min} = 1$ bridge$/5$ days, $s_{max} = 1$ bridge$/1$ days. For example, if $N = 1000$, then $T_0 = 75$ and $T_1 = 375$ days. f is the probability that a new invited user is malicious. The probability of blocking a bridge is p.

b) Minimum values: Here, we have some flexibility. Let's assume each user receives $k = 1$ bridge, and a bridge can be shared by $g = 5$ users. Initially, we recruit a great bunch of bridges, say one bridge in one day, and after stable, we recruit one bridge in five days; thus, $s_{max} = 1$ and $s_{min} = 0.2$. Now say we want to run the experiment for $T_1 = 30$ days, then $T_0 = 6$ days. Using this setup, we need $N = 6 * 5 * 1 = 30$ users and $\frac{35*1}{5} = 7$ bridges.

B. TorBricks

a) Protocol: n number of clients. t number of corrupt users. The Algorithm runs as $O(3 \lg n)$ instances in parallel. Total number of bridges required $(10t + 96) \lg n = O(t \lg n)$. Maximum rounds required to get each honest user have an unblocked bridge is $\lg(\frac{t+1}{32})$+1.

b) Minimum values: The underlying protocol assumes a fixed number of honest users and a fixed number of corrupt users. The corrupt users can help censor to block the received bridges either in minimum (prudent) or all (aggressive) or probability (stochastic) number of bridges. For our practical experiment, we consider real adversary and the number of corrupt users as unknown; hence we have $t = 0$ number of corrupt users and $96 \lg n$ number of bridges required. So if we have $n = 2$ users, the protocol needs 96 bridges, which is enormous in number, to tackle possible adversarial attacks.

C. Enemy At the Gateways

a) Protocol: Birth interval is 365 days. μ is the rate of new clients per time unit. μ_b is the rate during the birth interval, and μ_s is the rate after the birth interval. λ is the rate of new proxies per time unit. Similarly, λ_b is the rate during the birth interval, and λ_s is the rate after that. When adding a new user to the system, she will be a censoring agent with the probability ρ, otherwise a benign client. At a time, 3 proxies to each user, and the maximum 40 clients can be served by one proxy. $\mu_b = 25, \lambda_b = 5, \mu_s = 10, \lambda_s = 0.5, \rho = 0.05$

b) Minimum values: We can consider each user gets one bridge and one bridge can be shared by five users. The birth interval is 6 days and 1 day as a unit. We have $\mu_b = 1, \lambda_b = 1, \mu_s = 0.5, \lambda_s = 0.5$. If we run the protocol for 30 days with this setup, the total number of users and bridges will be 18 and $\frac{6*1+24*0.5}{5} = 4$ respectively.

VII. Evaluation plan

Currently, Tor distributes bridges via email and CAPTCHA. If we want to evaluate the bridge distribution protocols in the wild, we must modify the Tor Bridge distributor. In the following pseudocode, Algorithm 1, we design how the bridge distributor can work without providing privacy to the user similar to the current status of the bridge distribution.

In the 2018 Tor Blog post [8], Mike Perry discussed what can be a natural means to distribute bridge addresses after significant cloud infrastructure providers have blocked domain fronting. As a potential alternative, he considered the social-based cryptographic distributors like rBridge [9] and Hyphae [5] can be employed, and a practical evaluation can be performed to find the best mechanism. Our current project is aligned with this research direction, as well. Moreover, the feasibility study of a social bridge distributor started in 2012 [3], which finally triggered the implementation of Hyphae [4] in 2017.

Hyphae [5] is an advanced version of rBridge [9] mechanism which focused only on the privacy-preserving protocol of the social distribution. Notably, it uses keyed anonymous verification authentication instead of k-times anonymous authentication and drops Oblivious Transfer to keep the initial bridge distribution unlinkable. Other than these changes in a cryptographic protocol, Hyphae does some minor changes to the reputation system. The following pseudocode, Algorithm 2, for privacy-preserving bridge distribution is presented based on rBridge and Hyphae.

References

[1] Arun Dunna, Ciarán O'Brien, and Phillipa Gill. Analyzing china's blocking of unpublished tor bridges. In *8th USENIX Workshop on Free and Open Communications on the Internet (FOCI 18)*, Baltimore, MD, 2018. USENIX Association.

[2] David Fifield and Lynn Tsai. Censors' delay in blocking circumvention proxies. In *6th USENIX Workshop on Free and Open Communications on the Internet (FOCI 16)*, Austin, TX, 2016. USENIX Association.

[3] Isis Agora Lovecruft. Design a social bridge distributor. In *Tor Project Tickets*, 2012.

[4] Isis Agora Lovecruft. Implement the remaining cryptographic protocols for hyphae. In *Tor Project Tickets*, 2017.

[5] Isis Agora Lovecruft and Henry de Valence. Hyphae: Social secret sharing. As a Provisional Draft, 2017.

[6] Damon McCoy, Jose Andre Morales, and Kirill Levchenko. Proximax: Measurement-driven proxy dissemination (short paper). In *Proceedings of the 15th International Conference on Financial Cryptography and Data Security*, FC'11, pages 260–267, Berlin, Heidelberg, 2012. Springer-Verlag.

[7] Milad Nasr, Sadegh Farhang, Amir Houmansadr, and Jens Grossklags. Enemy at the gateways: Censorship-resilient proxy distribution using game theory. In *Proceedings of the Network and Distributed System Security Symposium*, NDSS'19, 2019.

[8] Mike Perry. Tor's open research topics: 2018 edition. In *Tor Project Blog*, 2018.

[9] Qiyan Wang, Nikita Borisov, and Zi Lin. rbridge: User reputation based tor bridge distribution with privacy preservation. In *Proceedings of the Network and Distributed System Security Symposium*, NDSS'13, 2013.

[10] Mahdi Zamani, Jared Saia, and Jedidiah Crandall. Torbricks: Blocking-resistant tor bridge distribution. In Paul Spirakis and Philippas Tsigas, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 426–440, Cham, 2017. Springer International Publishing.

Algorithm 1 Bridge Distributor

procedure BRIDGEDISTRIBUTION(user, token)
  if user is new and has a valid invite token **then**
    Check the IP address to verify if the user is new
    Create a credential token for the new user and mark it as used
    Choose a distribution mechanism uniformly at random
    Choose k unblocked bridges from the pool
    Create a new record for the user in the corresponding database of the mechanism
    For example, if the chosen mechanism is *rBridge*, the record contains user credential token, total credits of the user, for each assigned bridge user has bridge information, given time, and credits earned from the bridge.
    Send selected k bridges and the user credential to the user
  else if user is old and requests for a new bridge instead of blocked bridge B_b **then**
    Retrieve the record for the user using the submitted user credential token
    Check whether B_b is in the record and currently blocked
    Check if the user has enough credit to get a new bridge instead of B_b
    Modify the record (updated credit) for the user and send back a new bridge and updated credential token.
  else if user is old and requests for an invite token **then**
    Retrieve the record for the user using the submitted user credential token
    Check if the user has enough credit to get an invite token
    Modify the record (updated credit) for the user and send back the invite token.

Algorithm 2 Privacy-aware Bridge Distributor (detailed)

procedure PRIVATEBRIDGEDISTRIBUTION(user, token)
  if user is new and has an invite token **then**
    Verify the invite token and mark it as spent/used
    Create a credential token, wallet token with sufficient balance to buy k bridges, and k initial bridge tokens for the new user
    Choose a distribution mechanism uniformly at random
    Create a new record for the user in the corresponding database of the mechanism. For example, if the chosen mechanism is rBridge, the record's attributes are total credits, given time, blocked flag, and credits earned for each bridge.
    Send the credential token, wallet token, and k initial bridge tokens to the user
  else if user is old and requests for new bridge **then**
    Verify user's submitted credential, bridge token, and wallet transaction
    Retrieve and modify the record for the user by the credential token. For example, if the associated mechanism is rBridge, the record will have updated values in the attributes of total credits, given time for each bridge.
    Send a new wallet token (with updated credit), a new bridge, and new bridge token to the user
  else if user is old and earns reputation credit **then**
    Verify user's submitted credential, wallet token, and bridge token.
    Retrieve and modify the record for the user by the credential token. For example, if the associated mechanism is rBridge, the record will have updated values in the attributes of total credits and credit earned.
    Send a new bridge token (with an updated time-stamp) and new wallet token (with updated balance) to the user.
  else if user is old and reports a blocked bridge **then**
    Verify user's submitted credential, bridge token, and the bridge reachability
    Retrieve and modify the record for the user by the credential token. For example, if the associated mechanism is rBridge, the blocked flag of the bridge of the record will be correct.
    Update bridge database to mark it blocked
  else if user is old and requests for an invite token **then**
    Verify user's credential and wallet token if it is sufficient to invite other
    Retrieve and modify the record for the user by the credential token. For example, if the associated mechanism is rBridge, the record will have updated values in the user's total credits.
    Send new wallet token (updated balance) and blind-issued invite token to the user.