Enhancing the Unlinkability of Circuit-Based Anonymous Communications with *k*-Funnels

VÍTOR NUNES, INESC-ID / Instituto Superior Técnico, Universidade de Lisboa, Portugal
JOSÉ BRÁS, INESC-ID / Instituto Superior Técnico, Universidade de Lisboa, Portugal
AFONSO CARVALHO, INESC-ID / Instituto Superior Técnico, Universidade de Lisboa, Portugal
DIOGO BARRADAS, University of Waterloo, Canada
KEVIN GALLAGHER, NOVA LINCS, NOVA School of Science and Technology, Portugal
NUNO SANTOS, INESC-ID / Instituto Superior Técnico, Universidade de Lisboa, Portugal

Anonymous communication systems are essential tools for preserving privacy and freedom of expression. However, traffic analysis attacks make it challenging to maintain unlinkability in circuit-based anonymity networks like Tor, enabling adversaries to deanonymize communications. To address this problem, we introduce *k-funnel*, a new security primitive that enhances the unlinkability of circuit-based anonymity networks, and we present BriK, a Tor pluggable transport that implements *k*-funnels. *k*-Funnels offer *k*-anonymity to a group of *k* clients by jointly tunneling their circuits' traffic through a bridge while ensuring that the client-generated flows are indistinguishable. BriK incorporates several defense mechanisms against traffic analysis attacks, including traffic shaping schemes, synchronization protocols, and approaches for monitoring exposure to statistical disclosure attacks. Our evaluation shows that BriK is able to support web browsing and video streaming while offering *k*-anonymity. We evaluate the security of BriK against traffic correlation attacks leveraging state-of-the-art deep learning classifiers without considering auxiliary information and find it highly resistant. Although *k*-funnels require the cooperation of mutually trusted clients, limiting their coordination, our work presents a new practical solution to strengthen unlinkability in circuit-based anonymity systems.

CCS Concepts: • **Security and privacy → Usability in security and privacy**; **Privacy-preserving protocols**; **Pseudonymity, anonymity and untraceability**; • **Networks → Network privacy and anonymity**.

Additional Key Words and Phrases: k-anonymity, statistical disclosure, Tor, traffic analysis, unlinkability

ACM Reference Format:
Vítor Nunes, José Brás, Afonso Carvalho, Diogo Barradas, Kevin Gallagher, and Nuno Santos. 2023. Enhancing the Unlinkability of Circuit-Based Anonymous Communications with *k*-Funnels. *Proc. ACM Netw.* 1, CoNEXT3, Article 18 (December 2023), 26 pages. https://doi.org/10.1145/3629140

1 INTRODUCTION

Internet users are exposed to unprecedented levels of surveillance, tracking, and monitoring by governments, corporations, and other third parties [70, 72]. To counter these threats to privacy and freedom of expression, anonymous communication systems have emerged as central tools for protecting users' identities and ensuring their online activities remain private and secure [27, 75]. A

Authors' addresses: Vítor Nunes, vitor.sobrinho.nunes@tecnico.ulisboa.pt, INESC-ID / Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal; José Brás, jose.bras@tecnico.ulisboa.pt, INESC-ID / Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal; Afonso Carvalho, afonso.de.carvalho@tecnico.ulisboa.pt, INESC-ID / Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal; Diogo Barradas, diogo.barradas@uwaterloo.ca, University of Waterloo, Waterloo, ON, Canada; Kevin Gallagher, k.gallagher@fct.unl.pt, NOVA LINCS, NOVA School of Science and Technology, Lisbon, Portugal; Nuno Santos, nuno.m.santos@tecnico.ulisboa.pt, INESC-ID / Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal.

2834-5509/2023/12-ART18
https://doi.org/10.1145/3629140

crucial desirable property of these tools is *unlinkability*, which ensures that an eavesdropping adversary cannot link the source with the destination of communications, offering strong anonymity [43].

However, given the advances in traffic analysis attacks, unlinkability has become increasingly difficult to preserve in connection-based anonymity networks [75] such as the Tor [23] and I2P [36] networks. Circuit-based (also known as connection-based) communication is a latency-sensitive paradigm in which the sender and receiver establish a long-lived, encrypted, and multi-hop virtual circuit before transmitting or receiving data. For instance, in Tor, circuits are designed to tunnel TCP/IP streams, which require synchronous low-latency channels to achieve good performance.

Since circuits rely on long-lived connections between a few hops in the network, each of a circuit's packets cannot be sent through random independent paths. Unfortunately, this makes circuit-based communication substantially prone to traffic correlation [60, 66], website fingerprinting [16, 69], or statistical disclosure [19, 57] attacks. As shown in prior work [21], achieving strong security, low latency, and high throughput is impossible; only two of these are achievable simultaneously.

Using Tor as reference, we aim to explore a new design in circuit-based anonymity networks offering stronger security in exchange for lower bandwidth. We wish to ensure the unlinkability of circuits, making them resilient to advanced state-of-the-art machine learning based traffic analysis attacks such as DeepCoFFEA, and to statistical disclosure attacks (sans auxiliary information). In addition, the latency of the channels must remain low to ensure that TCP/IP streams can function, albeit we can tolerate a reduction in the channel's bandwidth so long as end-users can still access the Internet for typical tasks such as web browsing and video streaming.

To this end, we propose a new security primitive named *k-funnel* to enhance the unlinkability of circuit-based anonymity networks. Our primitive achieves this by offering k-anonymity to a group of k clients that can jointly tunnel their circuits' traffic through a common proxy while ensuring that the characteristics of individual client-generated flows are indistinguishable from one another. By doing so, we prevent an adversary from determining the true source of the traffic exiting the proxy thus breaking the linkability between the connected k clients and the observed destinations. Although k-anonymity has been successfully adopted in the past for message-based anonymity systems [35, 38, 93], implementing it in general-purpose circuit-based systems poses non-trivial challenges due to the tight low-latency and synchronization restrictions that need to be satisfied.

To demonstrate the feasibility and effectiveness of this primitive, we designed and implemented BriK, a system that enables the establishment of k-funnels as a pluggable transport for Tor. Released as open-source [65], BriK implements a bridge that serves as a k-funnel proxy for Tor clients. This bridge allows groups of k users to connect to the Tor network via k-funnels, offering k-anonymity for Tor users without requiring changes to the Tor protocols or software. BriK incorporates multiple defense mechanisms against traffic analysis attacks: i) special-purpose protocols that ensure synchronization between clients, bridges, and the Tor network, ii) traffic shaping techniques that guarantee indistinguishability of client connections, and iii) dedicated mechanisms for monitoring the exposure to statistical disclosure attacks and taking precautionary defensive measures.

We find that BriK offers sufficient bandwidth for web browsing and video streaming workloads while funneling the traffic of a few dozen clients through a single bridge. Though throughput decreased due to client synchronization, clients across nine residential networks averaged 1–3Mbps when funneling Tor circuits. Our evaluation revealed that BriK is resistant to the state-of-the-art traffic correlation attack DeepCoFFEA [66] aimed at breaking the unlinkability of circuits, even if subjected to active network manipulations. We also analyzed the success of statistical disclosure attacks in the absence of auxiliary information, which depend on how stable is the group of users who simultaneously funnel traffic through a bridge. Our evaluation shows that BriK can prevent statistical disclosure attacks by monitoring and, if necessary, reacting to users' browsing habits.

Our work represents a new practical application of *k*-anonymity to secure circuit-based anonymity systems, laying the groundwork for future improvements. We believe BriK can be especially useful for smaller user groups subject to mass or targeted surveillance such as investigative journalists fearing retaliation or oppression. By working together, *k*-anonymity can protect them individually, enabling them to form pockets of trusted peers and collaborators. To improve *k*-funnel connectivity opportunities, BriK hub devices can be deployed by fellow journalists and permanently connected to the Internet, in their homes or offices. BriK bridges can be deployed by trusted third parties, such as NGOs, press freedom organizations, and media outlets, to help protect investigative journalists' freedom, e.g., Amnesty International, Human Rights Watch, Reporters Without Borders, the Associated Press, the Guardian Project, the Freedom of the Press Foundation, and/or the International Consortium of Investigative Journalists. By leveraging the support of these organizations, BriK can offer *k*-anonymity to journalists and their sources.

2 BACKGROUND AND MOTIVATION

2.1 Circuits Under Traffic Analysis Attacks

Tor establishes encrypted channels called *circuits* through which user traffic is tunneled via a network of servers called *relays*. Once established, circuits can transmit latency-sensitive TCP/IP streams, encoding the packets inside fixed-side cells. The routing protocol ensures that user traffic is routed through three geographically-dispersed relays – *guard* (or *entry*), *middle*, and *exit* – to break the linkability between the source and destination of communications.

However, traffic analysis techniques can compromise unlinkability. As remarked by Tor's designers [82], this limitation is structural to Tor and is inherent to synchronous low-latency anonymity networks as the relays must forward payload packets as fast as possible to offer high performance and good web browsing experience. As such, emerging traffic patterns become hard to masquerade and can be detected through different approaches depending on the adversary's capabilities.

Two notable techniques include *traffic correlation* and *website fingerprinting* attacks. In traffic correlation [40, 79], the adversary must be able to monitor the guard and exit relays. Because observable traffic features such as the volume of inter-packet arrival times will be related between the flows near the source and the destination, the adversary can employ statistical analysis techniques [60, 61] to correlate flow pairs and break their unlinkability. In website fingerprinting [16, 69], the adversary only typically needs to monitor the user traffic at the guard and match the observed patterns against a collection of known website fingerprints. If a match exists, the adversary can establish a link between the source and the destination website, again deanonymizing the communication.

2.2 Unlinkability Through *k*-Anonymity

Our work aims to address the research gap in mitigating traffic analysis attacks and improving *unlinkability* in circuit-based networks like Tor. Borrowing the definition from Hopper et al. [38], by unlinkability we refer to the property that messages output by the anonymity network to a destination within a certain timeframe cannot be traced back to the messages that were originally entered into the network from a source during that same timeframe.

Existing protections attempt to avoid network regions controlled by an adversary: some strategies avoid potentially compromised relays [80], while others circumvent untrusted ASes [2, 26, 64, 76] or geographical regions [42, 53]. However, these strategies are ineffective if the adversary can still manage to intercept the traffic, e.g., by establishing new collusion agreements with specific ISPs.

To strengthen the unlinkability of circuit-based systems, we propose loosening the strict binding between the source and destination in circuits with *k*-anonymity. Each of the *k* clients has an equal probability (sans auxiliary information) of being the source of a flow targeting some destination,

thereby enhancing unlinkability proportionally to the group size k. To be effective, the adversary cannot distinguish individual participants based on their traffic features. Our approach must tackle non-trivial requirements due to the circuits' synchronized and low-latency nature:

- **R1. The k participants must be synchronously online:** Contrary to message-based paradigms, where participants can buffer and send messages asynchronously, circuit-based communications require all k participants to generate traffic simultaneously to cover for each other's network accesses. This restriction creates usability and availability hurdles, forcing participants to be online or to trust an online surrogate operating on their behalf.
- **R2. The k participants must generate indistinguishable traffic:** Unlike message-based paradigms, the strategies to achieve this property are more constrained in circuit-based communications as they cannot arbitrarily delay or mix packets. Even if padding and traffic modulation are employed to streamline the traffic characteristics, subtle differences can still emerge due to changing network conditions or the deliberate interference of an adversary. For instance, the adversary may selectively drop participants' packets and detect interruptions in the transmission to the destination, giving away the true source. Moreover, the traffic obfuscation mechanisms should not significantly reduce the throughput of the circuits or require excessive additional bandwidth resources to obfuscate communications, rendering the solution impractical.
- **R3. The k participants must be protected against statistical disclosure attacks assuming reasonable up-time:** A k-anonymity system should be able to resist statistical disclosure attacks, which arise when the k group of users changes over time. This variation in access patterns enables an adversary to collect observations that depend on the presence or absence of a particular participant in a session, potentially revealing the identity of the user. While prior systems [35, 38, 93] have explored solutions for this problem in message-based anonymity settings, it is crucial to investigate how building a K-anonymous circuit-based connection can make participants vulnerable to these attacks and devise appropriate measures to mitigate them.

It is worth noting that the existence of auxiliary information may make these attacks far more effective, depending on the auxiliary information that the adversary may have. For this work we assume that the adversary does not have such auxiliary information, and must resort to attacks based only on traffic analysis. We discuss this assumption more in Section 6.

3 OVERVIEW

This section introduces BriK, a system that enhances Tor circuits with k-anonymity. We present the system model and describe our threat model and assumptions.

3.1 System Model

Architecturally conceived as a Tor pluggable transport, the system model for BriK is centered on providing Tor users with k-anonymity through a new anonymization primitive called "k-funnel". This primitive allows communities of Tor users with k participants to create coordinated encrypted streams between a local *hub* and a dedicated *bridge* hosted by a trusted third party. Using these streams, participants can establish Tor circuits and tunnel their payload traffic through the bridge to the Tor network. By doing so, the source of the traffic exiting the bridge cannot be determined by an adversary since any of the k participants could be the likely source of a given circuit. Thus, the use of a k-funnel ensures the unlinkability between the source and destination of Tor communications.

In Figure 1, we illustrate this idea for a simple scenario involving three users. The hub, acting as a local gateway, provides a k-anonymous tunneling service for the circuits initiated by the Tor client software running on the users' computers. The hub service can run on a dedicated device or as a background process on the users' workstation or mobile phone. On the other hand, the bridge

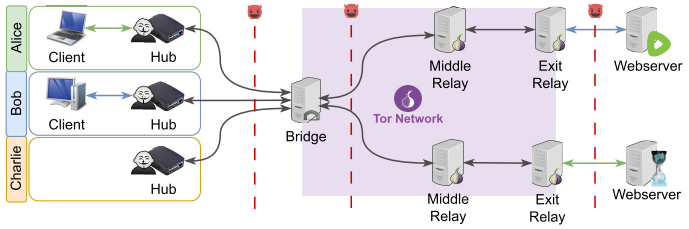


Fig. 1. BriK *k*-funnel where $k = 3$. An adversary can observe all flows indicated by the dashed red line. Alice and Bob access WikiLeaks and Rumble websites through independent Tor circuits. Charlie is offline.

is a standalone server that accepts connections from hubs. Both hub and bridge implement the Tor pluggable transport specification and are fully compatible with existing Tor protocols and software.

In this example, BriK enables three users to establish a *k*-funnel with k=3 to access Tor with *k*-anonymity properties. The *k*–funnel connects *k* hubs to the bridge, which serves as the Tor entry relay, and tunnels the Tor circuit traffic created by the participants through these connections, which we call *funnel streams*. In this case, Alice and Bob are accessing the Web through independent Tor circuits, with Alice accessing WikiLeaks and Bob accessing Rumble. Charlie is not currently online but has left his hub device online to act as his proxy and fulfill requirement R1 (see §2.2).

To make the streams indistinguishable (as per requirement R2), BriK encrypts and modulates them using a common traffic shaping function, generating fixed-sized packets at a pre-defined transmission rate. By doing so, an adversary will not be able to discern which stream carries the cell packets of each Tor circuit exchanged by their associated participants. In situations where there is no payload data to send, the hub and bridge may generate and transmit chaff to keep the streams active. In the case of Charlie's hub, which is online but not in use, it continually sends chaff.

3.2 Usage Model

Intended usage scenarios: BriK is not intended for all Tor users or to protect communities facing highly oppressive regimes with power and incentive to block Tor or arrest whole groups. Instead, we envisioned BriK for smaller user groups in democratic countries that may be the target of mass surveillance or endure political pressure, e.g., journalists. Fears of exposure may stifle their freedom to access/share sensitive information. BriK enables them to form pockets of trusted peers where they can benefit from *k*-anonymity, e.g., co-reporters from the same media outlet or from different news agencies. Adversaries (e.g., governmental agencies) can still identify the group but not trace individual web browsing patterns, protecting members from reprisals (e.g., being sacked).

Deployment and user management: BriK is meant to be deployed independently by a trusted organization acting as a service provider, such as a news agency. The service providers maintain one or multiple hubs and bridges permanently connected to the Internet. Some hubs might be set up within the provider's premises (enterprise hubs), while others could be placed off premises e.g., in users' homes (personal hubs). Bridges can be hosted on the organization's own network or in the cloud. Networking costs for hubs and bridges can either be borne by the owner or subsidized by the provider. Regardless of the hub deployment setup, the service providers ensure that only authorized entities can connect to their hubs. Users trust the service provider to identify and approve other participants thus ensuring these are legitimate. A user may be the only one accessing a hub at a given time or they might be sharing that hub with other users simultaneously (in enterprise hubs).

3.3 Threat Model and Assumptions

We consider an adversary that aims to break the unlinkability of a BriK *k*-funnel by determining which of the participating hubs hides the true source of a given Tor circuit exiting the bridge. From

the egress circuits of the bridge, we assume the adversary can determine the final destination (see §2.1): with global or state-level capabilities, it may deploy *traffic correlation attacks* by monitoring the traffic between the bridge and downstream middle nodes, and at the exit nodes; with local access to the bridge traffic, it can perform *website fingerprinting* on the traffic exchanged between the bridge and the Tor network (but should be unable to track a specific website access back to the client due to BriK's traffic shaping). We assume the adversary does not consider auxiliary data that could aid them in deanonymizing clients [30, 52, 56] using BriK, such as websites' kind, language, and typical access times [32], the physical location of visited websites and the location from which communications take place [20], or the identification of linguistic patterns of clients' posts to websites [4, 8, 32]. Still, for many BriK users (such as journalist organizations or activist/NGO coalitions), BriK bridges will be formed based on mutual interests and purposes, thus potentially mitigating the applicability of auxiliary data in traffic analysis to some extent.

The adversary can launch passive attacks (i.e., eavesdropping) and active attacks (i.e., selectively drop, modify, or inject packets) on this traffic. Although active attacks could trivially block availability to the bridge, the adversary's goal is to break the unlinkability between funnel streams and bridge egress circuits thus defeating k-anonymity. Akin to other pluggable transport implementations for the Tor ecosystem [6, 25, 81], defending the BriK bridge against DoS attacks constitutes a complementary challenge that we do not address in this paper. We cover the four following main attacks; each attack type leverages a distinct technique and, consequently, necessitates specific defensive mechanisms, which act as a roadmap for presenting our system in §4.3 and onwards.

- *Synchronization attacks:* Detect coordination discrepancies between hubs and bridge when managing k-funnels, e.g., the adversary can trivially link a participant's traffic if this participant is allowed to create a Tor circuit before the other k-1 participants have joined the k-funnel.
- *Traffic confirmation attacks:* Correlate the Tor circuit traffic flowing through the bridge with any of the k funnel streams by analyzing distinctive traffic features, e.g., volume of traffic, inter-packet arrival time, or packet lengths.
- *Statistical disclosure attacks:* Observe multiple funneled connections over time toward revealing BriK's clients communication patterns with their likely Internet destinations.
- *Bridge hijacking attacks:* Gain system administration privileges on BriK's bridge infrastructure and leak sensitive information or deviate from BriK protocols.

In addition to DoS attacks, we do not consider micro-architectural side-channel attacks on the bridge or attacks that exploit vulnerabilities in either the hub or bridge software. The threat of malicious hub operators is also not in scope; we assume that hubs and the bridge belong to BriK's trusted computing base. Furthermore, we assume that the participants of a k community of users are mutually trusted to actively follow the BriK protocols, do not collude with the adversary, exchange initial public key credentials out-of-band, and have access to the public key of an available bridge.

4 BRIK

This section presents the design of BriK. First, we give an overview of our system's architecture and describe the system's operation. Then, we detail BriK's key defense mechanisms.

4.1 Architecture and Implementation

Figure 2 depicts the internals of our system. BriK is a pluggable transport for Tor that comprises two components: the *hub*, which runs on the client-side, and the *bridge*, which runs on the server-side. Pluggable transports allow Tor traffic to be shaped and routed between clients and bridges without requiring changes to the Tor core. The hub and the bridge run independent BriK processes and implement the Pluggable Transport API [78] to interface with Tor processes running on the client and the bridge itself. This API allows setting up these processes to be used as a custom transport

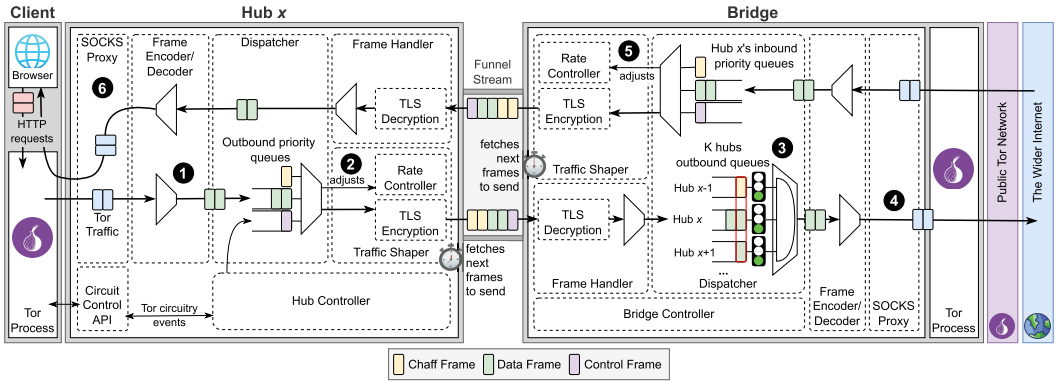


Fig. 2. Internal components of BriK and respective interactions: thick arrows represent the tunnel data path.

protocol for the Tor network. Hub and bridge also implement a SOCKS proxy interface which allows the hub to exchange data with a client application (e.g., a browser) via a local Tor process and the bridge to relay the traffic through the Tor network. In addition, the hub uses the Circuit Control API [84], to control the circuit-building process of the local Tor process. It allows the hub to intercept various circuit events, such as when a new circuit is created or when data is received on a circuit, and modify the normal behavior to restrict the creation of Tor circuits until the *k*-funnel is established or regulate the transmission of the circuits' cells.

Internally, hub and bridge include four main components which are fundamental for the deployment and operation of *k*-funnels: the *frame encoder and decoder*, the *dispatcher*, the *traffic shaper*, and the *frame handler*. To introduce these components, we explain how they serve the requests of a Tor client application to access the Tor network, assuming that a funnel stream has already been established as part of a currently active *k*-funnel with other hubs.

When a user initiates a web request in the browser, the local Tor client process needs to establish a circuit to tunnel the request. The hub component of BriK receives the notification of this event through the Circuit Control API. It first checks that a secure funnel stream with the bridge is already active before proceeding to create a new Tor circuit (see §4.2). Once the circuit is created, the client's Tor cells enter the hub through the SOCKS proxy and are encoded into fixed-size BriK frames by the frame encoder (step 1). The encoded frames are then forwarded to an outbound priority queue managed by the dispatcher. The traffic shaper, which contributes to mitigating traffic confirmation attacks (see §4.4), pops a frame from the queue at a fixed rate defined by the bridge (step 2). The frame is then sent to the bridge through an encrypted TLS channel.

On the server side, upon receiving data frames from a hub, the bridge decrypts them with the help of the frame handler and enqueues them in the outbound queue for the corresponding hub (step 3). When a frame has been received from every hub, as indicated by the green light in the traffic light symbol for each queue, the frame encoder dequeues one frame from each queue and processes it to relay the original request to the Tor network. This mechanism is one of BriK's defenses against traffic confirmation attacks (see §4.4). The frame decoder then decodes the data frames to obtain the original Tor cells and sends them to the SOCKS proxy interface living in the bridge to be forwarded to the Tor network (step 4). Replies from a client's destination are propagated back to the client by following the same steps in reverse, passing through the bridge (step 5) and then the hub (step 6).

In §4.5, we describe how BriK defends against statistical disclosure attacks. To shield BriK against bridge hijacking attacks, we leverage Intel SGX enclaves to protect the internal state of the bridge. We provide further details about our implementation and evaluation of this solution in Appendix A.

We implemented a BriK prototype for Linux. The source code is public [65]. We wrote the hub and bridge software components in ≈8 000 lines of C++ code using the OpenSSL and Boost libraries. We used C++ for increased performance. All components (including the circuit control API, in the case of BriK's hubs) are included as part of the same program, i.e., as part of a single binary. BriK's use of the PT API requires minimal changes to Tor and facilitates portability to different circuit-based anonymity networks as long as these support the PT API.

4.2 *k*-Funnel Setup

To create a *k*-funnel, the bridge first needs to know which hub devices can join the group. These devices must be registered and form the pool of potential participants. For authentication, each hub generates a public-private key pair, with the private key used to authenticate the hub towards the bridge. The bridge offers an HTTPS API that allows users to create groups and invite participants by uploading their respective hub public keys. The bridge also exposes a public key certificate that the hubs use for server-side authentication when initiating a connection to the bridge.

A *k*-funnel can be initiated in two ways once a pre-defined set of hubs agree to participate in said *k*-funnel. One way is to schedule rounds for specific dates and times, with a certain duration, during which the *k*-funnel is open. For example, a round may be open for 30 minutes and occur every other hour, starting at noon. When the hubs first connect to the bridge, they download the schedule and set timers for each round. Alternatively, a *k*-funnel can be initiated on demand by a specific member, with the request propagated to other members. Once the minimum required number of members agrees to participate, the round is scheduled and executed.

The *k* value for a *k*-funnel is determined by the BriK bridge when a set of participant restrictions is met. Each participant can set a lower bound *k-min*, specifying the minimum number of members required in the *k*-funnel. For example, Alice may set $k\text{-}min = 3$ indicating that she only wants to join *k*-funnels with at least three users, i.e., $k \geq k\text{-}min$. When a round starts, the bridge will maintain a waiting line of connecting hubs until all *k-min* restrictions are met, forming the *k*-funnel. Once all three funnel streams are ready, the hubs can authorize the local user to access Tor. Each participant can enhance their anonymity by increasing the *k-min*, potentially reducing their usability window.

4.3 Thwarting Synchronization Attacks

Synchronization attacks aim to deanonymize the creator of a Tor circuit by exploiting deficient coordination between hubs and the bridge during the formation and dissolution of *k*-funnels, as well as the creation and termination of Tor circuits. To mitigate these attacks, BriK implements a protocol between the hubs and the bridge (over a secure TLS channel) ensuring that certain invariants are preserved to maintain proper synchronization of these events.

Synchronization invariants: Three invariants must hold to coordinate *k*-funnel management:

- I1 *Funneling invariant:* Traffic from Tor clients must be tunneled exclusively through a *k*-funnel via their local BriK hub to prevent detection of the circuit's source.
- I2 *k-min invariant:* Tor circuits require hubs to meet the minimum *k*-min value specified by each user to be created and maintained. This ensures *k*-anonymity, where the probability of guessing the author of a Tor circuit is at most $1/k_p$ (k_p is the *k*-min restriction specified by any user).
- I3 *Fixed-set invariant:* Maintaining the set of hubs in a *k*-funnel fixed during a round is essential for ensuring *k*-anonymity. Any changes in the set of hubs, such as a new hub joining or an existing one leaving, require the termination of active Tor circuits and potentially their recreation. This is necessary to prevent an attacker from guessing the identity of a circuit's creator. Suppose a circuit is active in a *k*-funnel involving *k* hubs, and one hub leaves while the circuit remains alive. In that case, the circuit was not created by the client attached to the

exiting hub, increasing the probability of guessing the true creator to $1/(k-1)$. Similarly, if a new hub is allowed to join, guessing the circuit creator is skewed in favor of the adversary.

To enforce these invariants, hubs and bridges implement a specific network protocol. Next, we present this protocol and its state machine transitions from the hub's perspective (see Figure 3).

1. Joining a *k*-funnel: A hub starts in the `Init` state. Unless it joins a k-funnel, the hub does not allow the local Tor client to create new Tor circuits; this condition ensures I1. Whenever a round is scheduled to begin, the hub connects to the bridge (over an encrypted and traffic-shaped channel), and sends a `HELLO` message to join. This message contains the local user's k-min value, which indicates the minimum size of the set of online hubs demanded by the user before Tor circuits can be created (e.g., 3 in the example of Figure 1). In the first case, the hub enters a `Connected (Conn.)` state, otherwise, it enters `Shutdown (Shut.)` state disconnecting itself from the bridge. In the `Conn.` state, the hub can request the creation of Tor circuits if instructed by the local Tor client.

2. Creating a Tor circuit: When a connected hub intends to create a Tor circuit it will send a `CREATE` message to the bridge. The bridge will reply either with `ACTIVE`, placing the hub in the `Active` state and authorizing it to create the Tor circuit, or with `WAIT` otherwise. Receiving `WAIT` places the hub in the `Wait` state and tells it to suspend the Tor circuit creation until receiving an `ACTIVE` message. An `ACTIVE` will be sent by the bridge as soon as sufficient participants are online to satisfy each user's k-min, i.e., until the number of participants is greater or equal to the maximum of all k-min values of each participant. This ensures invariant I2. For instance, if k-min values of Alice and Bob are 2 and 3, they must wait until a new participant joins with k-min ≤ 3 for all of them to be able to create Tor circuits. At this point, every participating hub will receive an `ACTIVE` message and transition to the `Active` state.

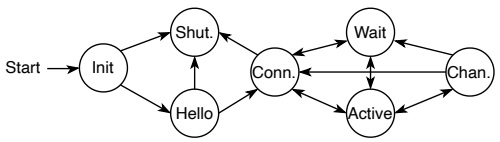


Fig. 3. State machine of a BriK hub.

3. Dealing with churn: An active hub can receive a `WAIT` message telling it that its k-min restriction does not hold anymore because one of the hubs has left and the current number of participants is less than the maximum k-min value required by the remaining participants in the circuit. Given that the hubs are dedicated devices stably connected to the Internet, we expect these situations to be relatively rare. Nevertheless, to guarantee I2 and I3, the hub reacts immediately by tearing down all its currently established Tor circuits. The hub will then need to wait until its k-min condition is again satisfied before new Tor circuits can be allowed. Apart from `WAIT` instructions, hubs can receive a `CHANGE` instruction which aims to preserve I3 if a participant has left but the k-min restrictions of the connected participants still hold. In this case, the bridge requests all connected hubs to tear-down existing circuits and create new ones. Upon sending a `WAIT` or `CHANGE` control frame, bridges immediately close active Tor client circuits and drop any data frames received after the control frame has been sent. In case of changing circuits, hubs should acknowledge bridges with a `CHANGE_OK` to resume the exchange of data frames. This prevents delivering Tor cells from previous Tor circuits (to the Tor network) after the client restriction has become invalid.

4.4 Thwarting Traffic Confirmation Attacks

To protect users against traffic confirmation attacks and preserve k-anonymity of a given k-funnel, the k streams interconnecting hubs and bridge must be *indistinguishable*, preventing an adversary from correlating individual streams with Tor circuit flows leaving the bridge or at exit relays. To achieve this, our solution involves dynamically shaping and throttling the funnel streams.

Traffic shaping: To mitigate traffic confirmation attacks, BriK encrypts and encodes all packets between hub and bridge inside *frames*. The size and sending rate of these frames are equalized by a

modulation signal across all streams of a k-funnel to make them indistinguishable. All frames in both directions have the same total length and provide a cover for tunneling Tor circuit cells and control plane messages. There are three types of frames (see Figure 2) whose format is detailed in Appendix B: *data frames* transmit Tor circuits' data, *control frames* exchange BriK protocol messages between hub and bridge, and *chaff frames* add noise and confuse an adversary [94].

To determine the cover signal to use for modulating traffic, there is a trade-off between network efficiency and performance. A bridge can specify different frame sending rate strategies:

- *Constant rate*: When the number of hubs is high, this strategy may be inefficient due to bandwidth waste, but has the benefit of giving a constant and predictable throughput.
- *Dynamic rate*: Calculated by dividing the maximum throughput of the bridge by the number of k hubs connected at a given time. Since this approach allows higher rates when the number of hubs is lower, the bridge can offer a better QoS to users. We opt to use this strategy by default.

At funnel setup time, the bridge sends the frame size and sending rate to the hub through the TLS channel. The traffic shaper dispatches the frames to the network at the agreed-upon sending rate. Control frames are given higher priority than data frames, enabling prompt delivery of control messages at the next traffic shaper event. If there are no data or control frames to send, a chaff frame is sent. The traffic shaper coordinates this process using a buffered chaff packet sample and two queues that buffer pending data and control frames. Figure 2 shows the outbound frame-sending mechanism in the hub (step 2) and the inbound frame-sending mechanism in the bridge (step 5) for the funnel stream between hub x and the bridge. The bridge maintains inbound queues for every k connected hub and iterates through them at each frame sending rate interval.[1]

Throughput leveling: Although hubs send traffic to the bridge at the same frame rate, variability in network conditions may lead to delays or bursts of frames. This can introduce characteristic timing differences in each funnel stream that can be reflected to the packets relayed to the Tor network, making passive attacks easier for the adversary. In addition, the adversary could actively drop frames of each funnel stream and detect interruptions in transmission of packets to the Tor network, thus identifying their source. To address this issue, we propose an adaptive *throughput leveling* mechanism, which ensures that the bridge only delivers traffic to the Tor network after receiving a BriK frame from all connected hubs, thus equalizing the throughput across all streams of a k-funnel. For example, two Tor cells received from different hubs within the same k-funnel with different uplink capacity are drained to the Tor network at the speed of the slowest client, preventing correlation between a hub and its Tor circuit based on timing properties.

This throughput leveling mechanism is illustrated in step 3 of Figure 2 and works as follows. The bridge maintains an outbound queue for each connected hub. When a frame is received from a hub, it is added to the corresponding queue. The bridge then waits until every queue holds at least one frame from each hub. Then, the dispatcher processes the frame at the head of the queue. If it is

[1]**Setting up the traffic shaper:** The *frame sending rate* (`FSRate`) and *frame size* (`FSize`) must be chosen carefully by the bridge operator as they impact network efficiency and performance. These parameters condition the *traffic shaping rate* (`TSRate`), which is the total throughput achievable by each funnel stream. We suggest an empirical approach: (1) The bridge operator picks the maximum bandwidth offered by the bridge (`bridgeCapacity`) and the maximum number of hubs connected simultaneously (`maxHubs`). These determine the minimum traffic shaping rate (`TSRate`) available to each hub; e.g., with `bridgeCapacity` = 125 Mbps and `maxHubs` = 25, `TSRate` = 5 Mbps when all hubs are connected. `TSRate` is dynamically adjusted based on the number of connected hubs (`nHubs`) as `TSRate` = `bridgeCapacity` / `nHubs`, providing smaller bandwidth to hubs as the number of connections increase. (2) The bridge operator then selects a candidate `FSize` which will determine the `FSRate` at which frames must be sent to meet the current `TSRate`. `FSRate` is calculated as `TSRate` / `FSize`. The frame size should be chosen such that it takes less time than `FSRate` to process and send frames to the clients to avoid timing differences between chaff and data frames. The value for `FSize` can be found experimentally by measuring the time it takes for the bridge to process and send a frame. (3) Following the procedure in (2), the bridge operator should test different `FSize` values. Depending on context, it may be better to leverage larger `FSize` and lower `FSRate`, or vice-versa.

a chaff frame, it is discarded. If it is a control frame, it is locally processed by the controller. If it is a data frame, it is dispatched to the Tor network. By ensuring that every frame sent to the Tor network is the result of at least one frame being received from each hub, this mechanism prevents an adversary from telling which hub is responsible for generating the packet relayed by the bridge.

Adaptability in response to nodes with variable bitrates: Whether due to network conditions or adversaries, our shaping mechanism only adjusts when a hub joins or departs from the bridge. As an illustration, let's consider a bridge with a capacity of 100Mbps, with two connected hubs each allowing a bandwidth of 50Mbps. When a third node joins the bridge, the shaping mechanism will proportionally adjust, reducing the bandwidth to approximately 33Mbps for each of the three hubs.

Alternative traffic shaping patterns: The current design of BriK leverages a naive traffic shaping mechanism which may be reminiscent of Herd [50], a system designed for anonymizing VoIP calls and which relies on constant rate and size chaffed traffic to conceal users' activities. We discuss the use of alternative (and potentially more efficient) traffic shaping patterns in Appendix C.

4.5 Thwarting Statistical Disclosure Attacks

Long-term observations of messages in an anonymity system enable an adversary to perform statistical disclosure attacks [19, 57], which probabilistically identify which participants access websites. They work by observing when a victim participates in an anonymity system, and which websites receive communication when the victim is online, accounting for some amount of "background noise" either estimated [19] or measured when the victim has not participated [57].

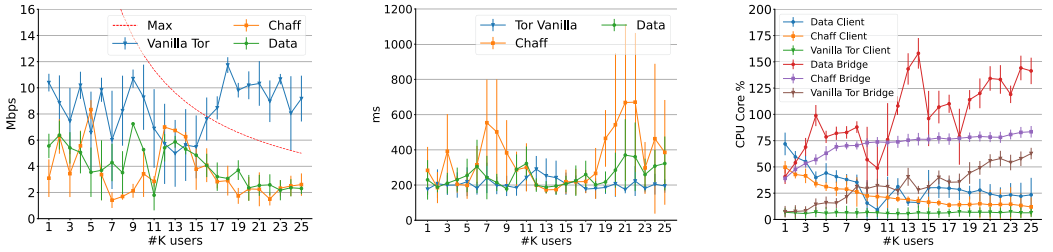
According to theoretical results by Hopper et al. [38], k-anonymity can prevent statistical disclosure attacks. In BriK, we rely on this property to protect users within a single k-funnel from such attacks. This is achieved through the k-min invariant and the fixed-group invariant (see §4.3), as well as traffic shaping and throughput leveling mechanisms (see §4.4), which ensure that each user's funnel stream is indistinguishable from others and that all users remain within a group of size $\geq k$. However, continuous observation by an attacker over multiple rounds of k-funnels may enable them to gain an advantage by discovering changes in the group membership of each k-funnel over time. We discuss potential additional mechanisms to avoid this in Appendix D. Note, however, that statistical disclosure attacks may be aided by auxiliary information that the adversary possesses. For this work we consider that the adversary has trivial amounts of information that does not aid in statistical disclosure, but the security of BriK against statistical disclosure attacks with auxiliary information is currently unknown. We discuss this issue more in Section 6.

5 EVALUATION

Our main evaluation goals are twofold: (i) assess the network performance of BriK, and (ii) measure its resistance against multiple attacks aimed at deanonymizing BriK users. Our evaluation assumes that each hub serves a single user, and that multiple hubs are connected to a single BriK bridge. Supporting multiple users per hub will ultimately result in the multiplexing of a hub's current `TSRate` bandwidth amongst its potential multiple end-users.

5.1 Performance in a Real-World Deployment

To mimic a set of users interested in leveraging BriK, we performed a practical deployment of BriK using 9 Raspberry Pi4 (RPis) devices acting as BriK hubs, each provisioned with a 4-core Broadcom BCM2711 CPU and 2GB of RAM. These devices were distributed among different members of our research group and installed in households across the metropolitan area of a major European city. This distribution enabled us to gather a representative sample of the ISPs and Internet connections' performance expected to be found in the homes of individuals interested in using BriK. Indeed, our



(a) Throughput. (b) Latency. (c) CPU consumption.

Fig. 4. Throughput, latency, and CPU consumption in the BriK baseline (`FSize` = 3125 B, `TSRate` = 5 Mbps).

measurements using `iperf3` reflect a heterogeneous landscape of network performances, where the raw throughput of RPi nodes (without using BriK or Tor) ranged from 17 Mbps to 203 Mbps. We see that even the low performance nodes achieve a sufficient throughput to sustain bandwidth-hungry applications, like video streaming. The full results of our measurement are detailed in Appendix E.

We then used our RPi deployment to run periodic BriK rounds based on a k-funnel (k=9) while using a traffic shaping rate of ≈5 Mbps, i.e., equivalent to 720p video streaming (following the traffic shaper setup process described in §4.4). Every two hours, the RPis engaged in a BriK round where device *rpi3* sent real data, in the form of an `iperf3` measurement, and all other hubs sent chaff. Our experiment revealed that BriK's traffic was rather stable, achieving a median throughput of 2.3 Mbps. (Figure 12 in Appendix E details our throughput measurements.)

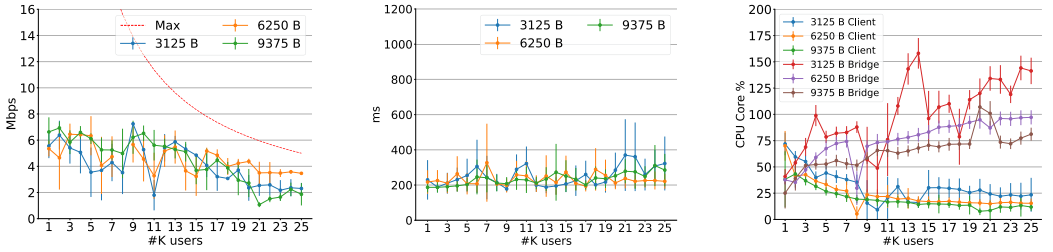
Further, we explored BriK's performance when hubs are deployed in a more stable environment, such as within well-connected enterprise-grade networks (e.g., as part of large organizations or news outlets). To this end, we deployed a BriK bridge (4 vCPU, 4GB RAM) in a Google Cloud instance located in Europe along with k=9 hubs split across the United States (East Coast), Western Europe, and India. We conducted a performance measurement in this setting while measuring throughput of one of the hubs located within the US. We observed a median throughput of ≈ 6.4Mbps, and a throughput of 7.9 and 4.2Mbps, on the 95th and 5th percentiles, respectively (detailed results in Appendix F). This suggests that BriK users may reap larger benefits by placing hubs within their organizations' headquarters rather than in their own home networks.

5.2 Baseline Microbenchmark Evaluation

We now evaluate the performance of a BriK baseline deployment. Our results suggest that a bridge with modest hardware (4vCPU, 4GB RAM) is able to support up to 25 BriK hubs (henceforth referred to as *clients* or *users*), while delivering enough performance for common Internet tasks.

Our testbed includes a bridge and multiple clients, where each node runs an instance of BriK (using Tor *v0.4.2.8*). Unless noted otherwise, we use a baseline configuration of BriK with a *traffic shaping rate* of 5 Mbps, a *frame size* of 3125 B, and a *frame sending rate* of 1.66ms. Thus, provided that the maximum number of hubs supported by a k-funnel are connected in a given round, each BriK hub will shape traffic such that each k-funnel stream will exchange traffic *at least* with a throughput of 5 Mbps. We chose a frame size of 3125B after following the traffic shaper setup method in §4.4. We explore other traffic shaper parameters in §5.3. Nodes run in Docker containers, with 4 vCPU and 4GB of RAM for the bridge node, and 2 vCPU and 1GB RAM for clients. Bridge and clients run in separate physical hosts with 2 Intel Xeon Silver 4114 vCPUs. Each client uses a Tor circuit as created by the Tor binary (i.e., we do not choose specific circuits for clients to use).

Throughput: Figure 4(a) depicts the average throughput obtained by a single BriK client over 10 runs according to an increasing number of users connected to the same k-funnel, both when additional clients exchange chaff and when additional clients exchange payload data. For comparison,



(a) Throughput (b) Latency (c) CPU consumption.

Fig. 5. Throughput and latency for different frame sizes. All clients exchange data (TSRate = 5 Mbps).

we show the results for an increasing number of users connected to the bridge using vanilla Tor. Note that relays may impose per-client bandwidth restrictions which are not made public.

The figure also shows that BriK clients generally achieve a throughput between 2Mbps and 7Mbps, shorter than the ≈6–12 Mbps achieved by vanilla Tor clients. When additional users connect to the same bridge, throughput degrades from ≈4-6 Mbps when $k \leq 10$ to ≈2Mbps when $k > 20$. This reduction is similar both when clients are receiving useful data and chaff. It is worth noticing that these results allude to the inherent variability of Tor circuits' performance, which transcends past the operation of BriK; for instance, vanilla Tor's throughput dipped when we experimented with a number of users between 11 and 17, before resuming a higher overall throughput for $k \geq 18$.

Latency: We measured the RTT of BriK and vanilla Tor channels by using the `httping` tool. Figure 4(b) depicts the average latency of 10 runs according to the increasing number of connected users k receiving chaff and k users receiving data through Tor. In general, having more users exchange payload or chaff through the BriK bridge (green and orange lines) causes the latency to increase. For a k-funnel where two users exchange data, the experienced latency is about 200ms, while latency sits around 400ms for the same type of k-funnel with more than 20 connected users. This latency sits close to that experienced by vanilla Tor users (200ms to 300ms).

Resource utilization: We measured the BriK bridge and clients' CPU usage as clients joined the system. For comparison, we performed similar measurements for a vanilla Tor bridge and its clients, where the bridge simply acts as an entry point to the Tor network, i.e., it does not perform traffic obfuscation. Figure 4(c) depicts clients and bridges CPU utilization. We can see that processing chaff frames is more lightweight than data frame processing. For 15 BriK users that simultaneously exchange data, the bridge shows an approximate usage of 1 core. This value decreases to about 0.75 cores when a single client (out of 15) exchanges data and the others exchange chaff. We also observe a trend which shows that CPU usage increases as more clients connect to the bridge, and that vanilla Tor bridges are ≈2.5× more efficient than BriK ones when 25 clients exchange data.

5.3 Varying BriK Traffic Shaping Parameters

Alternative frame sizes: We repeated our baseline experiments using a traffic shaping rate of 5 Mbps, but adapted our frame sending rate to use frames with double (6250 B) and triple (9375 B) the size of our baseline frame (3125 B). Figure 5(a) shows that these configurations do not lead to a significant reduction in throughput, except when $k \geq 18$, where the throughput of 9375 B halves when compared to 6250 B, and is similar to that of our initial frame size (3125 B). As in §5.2, this variability may also be explained by momentary congestion in the Tor network. Figure 5(b) depicts the latency obtained by BriK using different frame sizes. Our results reveal that the tested frame sizes do not cause a significant impact in the latency experimented by BriK users. For instance, for a large pool of users exchanging Tor traffic (k≥20), connections appear to experience the same latency either using the larger (9375B) or smaller (3125 B) frame sizes under test. Lastly, Figure 5(c)

shows the bridge CPU usage. Larger frames impose a smaller overhead in the bridge CPU, e.g., 0.75 cores for a 9375 B frame vs. 1.4 cores for a 3125 B frame when k=25 clients exchanged traffic.

Alternative traffic shaping rates: Apart from web-browsing, Tor users may wish to conduct other bandwidth-hungry activities privately, like video streaming. In this experiment, we configured BriK to use the same frame size (3125 B) but a different traffic shaping rate. Namely, we provide users with 3 or 5 Mbps traffic shaping rates for enabling 480p or 720p streaming [62], respectively.

To assess the quality of experience perceived by BriK users when streaming video, we measured the number of *frames per second* (FPS) transmitted by an HTTP-based video server while using the same BriK parameter configuration as above. We streamed a 30 FPS video via HTTP from a VLC video server (*v3.0.17.4*) hosted on Google App Engine. One user runs a VLC client instance while the two remaining users support the k-funnel with chaff traffic. In both cases, BriK closely matched the performance of vanilla Tor, fluctuating between ≈25 and 30 FPS (see Appendix G).

5.4 Resistance against Traffic Analysis

We now evaluate BriK's ability to resist the state-of-the-art traffic correlation attacks.

Classifiers: We used DeepCoFFEA [66], a state-of-the-art approach for end-to-end traffic correlation that relies on deep learning, to experiment with the correlation of k-funnel traffic (i.e., the traffic exchanged between BriK clients and the bridge) with HTTP/HTTPS traffic observed at the egress of the Tor network. To explore the success of a traffic analysis-capable adversary to distinguish users who are actively participating in the system (i.e., sending real data) from those that are simply supporting the k-funnel (i.e., sending chaff), we experimented with an XGBoost binary traffic classifier [11] paired with a comprehensive set of features based on statistics computed over packet lengths and inter-arrival times. We also adapted DeepCoFFEA to train on sets of ingress flows only.

Datasets: We built two representative datasets. The first dataset is used for assessing the indistinguishability between BriK payload and chaff traffic. The second dataset is used for evaluating BriK's resistance against traffic correlation. For generating both datasets, we deployed three clients connected to a BriK bridge, forming a k-funnel (k=3). In this setting, one client visits each of the Alexa top 20 000 websites in sequence, while the remaining clients support the k-funnel by simply sending chaff traffic. We collect the individual traffic flows pertaining to each website visit. We used `Selenium` with a Firefox driver coupled with Tor *v0.4.2.8* to fetch the pages as in a regular web-browsing session. We configure the client to forward traffic through a bridge controlled by us. Mimicking earlier testbeds [61], we capture the traffic generated by clients, and the traffic exchanged between the circuit's exit node and a proxy under our control.

Security metrics: To assess the effectiveness of traffic analysis attacks against BriK's k-funnels, we rely on the classifiers' *true positive rate* (TPR), *false positive rate* (FPR), and the *area under the ROC curve* (AUC), which depicts the trade-off between the TPR and the FPR. A perfect classifier obtains an AUC of 1, whereas an AUC of 0.5 is equivalent to random guessing.

Preventing traffic correlation: Figure 6 depicts the ROC curve obtained when using DeepCoFFEA to perform traffic correlation. The figure shows that vanilla Tor flows are highly susceptible to successful correlation (AUC of 0.9968) but that BriK flows, both at 3 and 5Mbps rates, cannot be accurately correlated (AUC≈ 0.5, i.e., close to random guessing). This suggests that an adversary is unable to link k-funnel streams carrying Tor traffic with the HTTP/S flows exiting Tor.

Preventing the identification of k-funnel traffic types: Similarly to the previous setting, our modified version of DeepCoFFEA achieves an AUC≈0.5 when attempting to distinguish whether a given client is exchanging BriK payload data or chaff. Our XGBoost binary traffic classifier obtained similar results to DeepCoFFEA (AUC≈0.5), suggesting that it is hard to distinguish data from chaff.

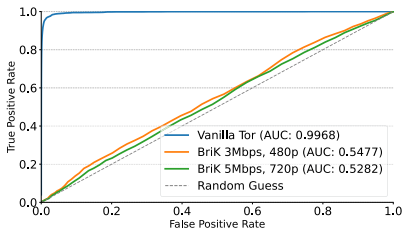
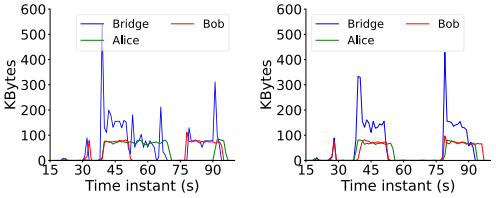


Fig. 6. DeepCoFFEA traffic correlation results (ROC).



(a) No throughput leveling (b) With throughput leveling

Fig. 7. Bytes received when dropping all packets from Bob, without and with throughput leveling.

Tolerating active network perturbations: An adversary could benefit from observing traffic patterns caused by unpredictable network events (e.g. full packet loss due to ISP outages) or active perturbations (e.g., intentional packet drops). Consider that Alice and Bob form a k-funnel with $k = 2$, and both start to upload a file over Tor. Then, during this operation, Bob's ISP suffers an outage and is unable to route Bob's packets towards the BriK bridge. Without a proper protection mechanism, an adversary could observe that the bridge continues to exchange traffic with the Tor network and unmistakably identify Alice as the source of the only active circuit leaving the bridge.

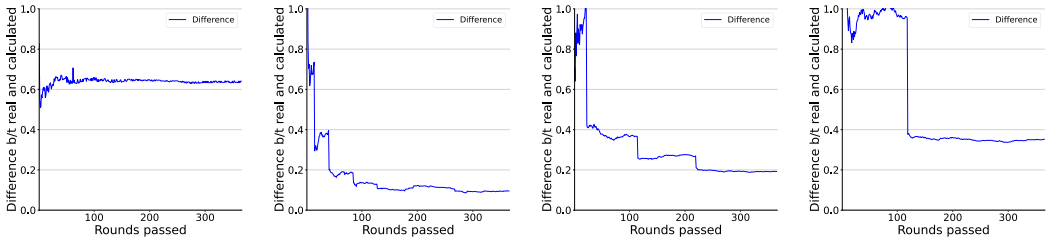
To protect against such attacks, we introduced a throughput leveling mechanism (§4.4) that stalls the traffic exchanged between clients within a k-funnel and the bridge upon detecting interference with BriK frames' sending rate. Figure 7 (a) and Figure 7 (b) depict the amount of bytes exchanged by both clients and the BriK bridge, without and with throughput leveling protection, respectively. The blue line shows the aggregate traffic from both Alice's and Bob's clients exiting the bridge to the Tor network. Suppose that Bob's ISP suffers an outage at $t = 51$s (we use `tc` to emulate total packet loss). Without the protection, the bridge continues to facilitate the exchange of Tor data with Alice. In turn, when applying the protection mechanism, an outage of Bob's ISP causes the bridge to *completely* stall its Tor data exchanges. Note that Alice keeps sending BriK frames to the bridge for a short period until she receives no response back. This however will not allow an adversary to pinpoint which client is connected to a given Tor circuit.

While the above example focused on full packet loss conditions for ease of exposition, our throughput leveling mechanism operates in a similar way when considering other (potentially adversarial) network perturbations (e.g., constrained bandwidth, added latency, etc.).

Attacks considering auxiliary information: While DeepCoFFEA works well against typical Tor traffic, it is not designed to break k-anonymity. Note, however, that BriK may not be secure against other attacks, specifically those that consider auxiliary information toward breaking k-anonymity (c.f. §3.3). We leave the investigation of such techniques, as well as how machine learning-based traffic analysis attacks may be modified to include such auxiliary information, to future work.

5.5 Resistance against Statistical Disclosure

Hopper et al. [38] showed that k-anonymity offered theoretical protection against statistical disclosure. To examine how BriK performs in unreliable environments, we measure its resistance to statistical disclosure under high-failure conditions. To do this we simulated an unlikely failure scenario for the BriK system – we assumed that a client's hub would drop out of communication and cause the users to drop below the k threshold, and that the BriK bridge would be unable to detect this and still allow clients to send traffic. As such, the word "failure" for a given user in our analysis means (1) the user does not send either chaff nor real traffic; and (2) for some unknown reason, the BriK system fails to pick up this user failure and decides to continue with communication anyway. As such, this experiment is a conservative estimate of how much statistical disclosure will



(a) SDA - failure rate of 10% and max. of 1 msg/round.

(b) ISDA - failure rate of 10% and max. of 100 msg/round.

(c) ISDA - failure rate of 1% and max. of 100 msg/round.

(d) ISDA - failure rate of 0.1% and max. of 100 msg/round.

Fig. 8. Results from statistical disclosure experiments.

affect our system, as we programmed the system to detect a lack of user participation and halt communication in the case that this occurs. We implemented two statistical disclosure attacks: the original attack [19] (*sda*), and the improved attack [57] (*isda*). We ran 30 attack simulations against generated BriK data. For each simulation we generated a different probability distribution per user. Consider a set of sites S that are accessible from a BriK bridge, and a set of user hubs U who are using BriK via the same bridge. Then, $\forall u \in U$ we generate $\vec{v}_u$ s.t. $\forall s \in S, v_s \in \vec{v}_u$ is a randomly chosen variable $v_s \in [0, 1]$ and $\sum_{s \in S} v_s = 1$.

We generate probabilities in this way because we believe it to be realistic, as each BriK user is likely to have different browsing habits. This method of probability generation directly violates the assumption presented in *sda* [19], where only one user, Alice, is assumed to have a unique probability distribution while others draw uniformly at random. As such we include *isda* [57], which does not assume the probability distribution of non-Alice users, but models the noise generated by them using data collected from rounds in which Alice fails.

Half of our simulations were performed using *sda*, and half using *isda*. For both attacks we ran simulations with differing probabilities of user failure, which remained constant for the duration of a simulation. The probabilities of failure were based on the five-nines: 90% availability (10% failure), 99% availability (1% failure), etc. User participate in every round for which they do not fail.

We ran experiments looking at the effect of the number of messages sent by users per round. Users sent a random number of messages with a set maximum, which remained constant during each simulation. For each failure probability we changed the parameter of maximum messages allowed to 1, 10, and 100 messages. Finally, we measure the success of statistical disclosure based on the average of the differences between a user's probability of visiting a given site and the adversary's approximation of it. Given a user's probability distribution $\vec{v}$ and the results of an attack $\vec{v\prime}$ where $|\vec{v}| = |\vec{v\prime}| = |S|$, we define the attack success for a round as: $d = \sum_{s \in S} abs(\vec{v}_s - \vec{v\prime}_s)/|S|$.

Figure 8(a) shows the result of *sda* against BriK assuming a failure rate of 10% and a maximum of one message per round. As predicted, *sda* is unsuccessful against BriK, likely due to our method of generating probabilities over S. All other results for *sda* are similar.

Isda provided different results. With a failure rate of 10% and a maximum of 100 messages per round, after 365 rounds *idsa* computed a probability distribution that was on average 0.09 away from the user's probability. With more rounds the adversary could approximate the correct probability with more accuracy. The results of this simulation are shown in Figure 8(b). When the user fails only 1% of the time with 100 maximum messages per round, the average difference on round 365 raises to 0.19, and when the user fails 0.1% of the time it raises to 0.35. These results are shown in Figures 8(c) and 8(d), respectively. In the case of 99.999% up-time, the attack fails entirely.

Thus, our solution is vulnerable to *isda* only under certain conditions. Given that our BriK deployment shows more than 99.9% up time, and that a user sends chaff whenever they are not

participating, we consider a real world deployment of BriK under high availability to be resistant to statistical disclosure attacks. This is consistent with the theoretical results of Hopper et al. [38]. Note, however, that this analysis assumes that the adversary does not have access to useful auxiliary data. We leave the exploration of the impact of auxiliary data on BriK's security to future work.

6 DISCUSSION

Privacy and usability trade-offs: To ensure *k*-anonymity, BriK must maintain a fixed set of hubs as part of a *k*-funnel while a round is in progress, requiring that Tor circuits can be terminated and re-created upon set membership changes, disrupting user experience and introducing a delay in service. By keeping clients independent from their hubs, a client can disconnect while keeping the respective hub active in chaff sending mode, thus avoiding the need for Tor circuit termination.

Hubs' connectivity: While the connectivity of hubs deployed in major cities (see Figure 12) can be more stable than in other regions of the world, our statistical disclosure experiments (§5.5) reveal that BriK is robust against such attacks even for large node failure rates (e.g., 90% availability). Ultimately, we expect the churn experienced by hubs to be dependent on the type of hub deployed. Enterprise hubs are expected to offer higher availability and to experience lower recovery times after failure than personal hubs.

Connecting users to remote hubs: Our prototype (§ 4.1) assumes users are co-located (i.e., within the same network) with the hub they wish to use to participate in the system. However, BriK's design does not preclude users from establishing secure connections to remote hubs. To reap the benefits of *k*-anonymity in such a setting, users must ensure that the traffic they exchange with a remote hub does not disclose the user's real online behavior via Tor. To this end, users may utilize existing traffic obfuscation techniques [12, 73, 77] to break the link between the traffic observed by a network adversary and a user's real traffic patterns. Investigating the performance penalties imposed by traffic obfuscation schemes on BriK remote hub access is left for future work.

Improving resistance to statistical disclosure: Though our results are promising in thwarting statistical disclosure, we did not analyze our system with respect to auxiliary data. However, the fact that BriK is envisioned to be used by organizations with similar interests may limit the impact of auxiliary data on these attacks, though we leave analyzing this claim to future work. Another direction to improve on BriK against statistical disclosure attacks is by using a statistical disclosure oracle and a policy to determine not only when a client should send chaff, but then mimicking a seemingly "normal" pattern of interactions with websites [54] (automatically clicking links, etc.).

DoS attacks: BriK slightly enlarges the attack surface for DoS attacks in that an adversary actively interfering with a single connection between a hub and a bridge, e.g., by dropping its packets, would automatically block the traffic of all other *k*-1 participant hubs in a *k*-funnel. This is an inherent limitation of our design and a price to pay to ensure unlinkability among the *k* participants.

Scalability: The BriK service can easily scale vertically (by increasing the compute resources of hubs and bridges to accommodate more hubs) and, perhaps, horizontally (by deploying additional bridges and by distributing the hubs' workloads among them). However, further research is required to assess the potential security implications of a dynamic distribution of hubs between bridges.

7 RELATED WORK

A popular use case for *pluggable transports* (PTs) [83] is to shape and obfuscate TLS flows carrying Tor traffic for bypassing Tor blocking measures [6, 25, 81] or foiling *website fingerprinting* [33, 34, 37]; for instance, similar in spirit to BuFLO [24], BriK also shapes Tor traffic into a constant-rate traffic stream. However, in contrast to BriK, these solutions offer no anonymity guarantees if an adversary can probe further into the Tor network egress links and perform statistical disclosure attacks.

To prevent traffic correlation threats, Tor clients may attempt to avoid unsafe relays [13, 22, 90, 91]. However, existing strategies have been found to leak information about clients' locations to adversaries, which can then link partial traffic observations [88] or to strategically place malicious relays [89]. CLAPS [71] aims to mitigate such issues. Yet, CLAPS assumes an adversary that may have access to some strategic network locations, but not all, as we assume in BriK. Traffic patterns are also not obfuscated by CLAPS in any way. AS-aware path selection algorithms help decrease the chance of an AS-level attacker observing traffic flowing between both endpoints of a Tor circuit [2, 26, 64, 76]. DeTor [53] and Trilaterator [42] present improved routing schemes to prevent Tor circuits from traversing specific regions but do not fully prevent traffic correlation efforts.

To thwart statistical disclosure, Buddies [93] groups users with a fixed set of peers that access some predetermined content at fixed intervals and in a round-robin fashion. TASP [35] aims to reduce Buddies' latency at the cost of an increased chance of deanonymization. However, both systems are only applicable to message-based anonymity networks. Aqua [51] provides k-anonymity on low-latency mix networks as long as the adversary colludes only with one end of the path between the communicating clients. These assumptions result in security concerns that are akin to those of Tor, as adversaries who observe both ends of the circuit can still deanonymize clients [67].

Many systems are based on various techniques to protect sensitive communication metadata from a global adversary: trusted hardware [39, 74], multiparty computation [1, 3, 28, 55], private information retrieval [5, 17, 29, 46, 86], mix networks [9, 10, 15, 44, 45, 68, 92], and differential privacy [47, 49, 85, 87]. Among these, specific works like Stadium [85], Karaoke [47], or Groove [10] leverage parallel mixnets to prevent global adversaries to correlate which users are communicating (and at what times), by pseudorandomly connecting users to dead drops (i.e., ephemeral addresses where users exchange messages) while introducing noise traffic that confuses an adversary's observations. BriK can be perceived as using similar ideas based on the introduction of chaff traffic to hide the correspondence between users and the online content they access via Tor.

However, most of the systems listed in the previous paragraph do not provide general support for low-latency communication like web-browsing. Instead, they are often focused on application-specific scenarios, such as message broadcasting [1, 17, 18, 63], private messaging [10, 28, 29, 47], file-sharing [31, 51, 86], VoIP [48, 50], or future Internet architectures [14, 15]. In these works, Tor is usually criticized for its vulnerability to global network adversaries and susceptibility to traffic confirmation attacks. BriK fills this gap, enabling Tor to resist traffic correlation and statistical disclosure, protecting users with k-anonymity. PriFi [9] supports traffic analysis-resistant and low-latency web-browsing within LANs, but is vulnerable to statistical disclosure attacks.

8 CONCLUSIONS

This paper introduced the k-funnel primitive for improving unlinkability in circuit-based anonymous systems and presented BriK, a new pluggable transport for Tor designed to make it robust against traffic correlation and statistical disclosure attacks. Even if mounted by an adversary that can intercept the traffic at every link of the Tor network, BriK can provide k-anonymity for groups of Tor users and implements efficient defenses against traffic analysis.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their comments and insightful feedback. This work was partially supported by the Fundação para a Ciência e Tecnologia (FCT) under grant UIDB/50021/2020 and by IAPMEI under grant C6632206063-00466847 (SmartRetail), by NSERC under grant RGPIN-2023-03304, and by NOVA LINCS (UIDB/04516/2020) with the financial support of FCT.IP.

REFERENCES

[1] Ittai Abraham, Benny Pinkas, and Avishay Yanai. 2020. Blinder–Scalable, Robust Anonymous Committed Broadcast. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 1233–1252.

[2] Masoud Akhoondi, Curtis Yu, and Harsha V. Madhyastha. 2012. LASTor: A low-latency AS-aware Tor client. In *Proceedings of the IEEE Symposium on Security and Privacy*.

[3] Nikolaos Alexopoulos, Aggelos Kiayias, Riivo Talviste, and Thomas Zacharias. 2017. MCMix: Anonymous Messaging via Secure Multiparty Computation. In *Proceedings of the USENIX Security Symposium*. 1217–1234.

[4] Mishari Almishari and Gene Tsudik. 2012. Exploring linkability of user reviews. In *Proceedings of the European Symposium on Research in Computer Security*. 307–324.

[5] Sebastian Angel and Srinath Setty. 2016. Unobservable Communication over Fully Untrusted Infrastructure. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*. 551–569.

[6] Yawning Angel. 2019. obfs4 (The obfourscator). https://github.com/Yawning/obfs4/blob/master/doc/obfs4-spec.txt.

[7] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'keeffe, Mark L Stillwell, et al. 2016. SCONE: Secure linux containers with intel SGX. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*. 689–703.

[8] Michael Backes, Pascal Berrang, Oana Goga, Krishna P Gummadi, and Praveen Manoharan. 2016. On profile linkability despite anonymity in social media systems. In *Proceedings of the ACM on Workshop on Privacy in the Electronic Society*. 25–35.

[9] Ludovic Barman, Italo Dacosta, Mahdi Zamani, Ennan Zhai, Apostolos Pyrgelis, Bryan Ford, Jean-Pierre Hubaux, and Joan Feigenbaum. 2020. PriFi: Low-latency anonymity for organizational networks. *Proceedings on Privacy Enhancing Technologies* 2020, 4 (2020).

[10] Ludovic Barman, Moshe Kol, David Lazar, Yossi Gilad, and Nickolai Zeldovich. 2022. Groove: Flexible Metadata-Private Messaging. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*. 735–750.

[11] Diogo Barradas, Nuno Santos, and Luís Rodrigues. 2018. Effective Detection of Multimedia Protocol Tunneling using Machine Learning. In *Proceedings of the USENIX Security Symposium*. 169–185.

[12] 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 ACM Conference on Computer and Communications Security*. 35–48.

[13] Sambuddho Chakravarty, Georgios Portokalidis, Michalis Polychronakis, and Angelos D. Keromytis. 2011. Detecting Traffic Snooping in Tor Using Decoys. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*. 222–241.

[14] Chen Chen, Daniele E Asoni, David Barrera, George Danezis, and Adrian Perrig. 2015. HORNET: High-speed onion routing at the network layer. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 1441–1454.

[15] Chen Chen, Daniele E Asoni, Adrian Perrig, David Barrera, George Danezis, and Carmela Troncoso. 2018. TARANET: Traffic-analysis resistant anonymity at the network layer. In *Proceedings of the IEEE European Symposium on Security and Privacy*. 137–152.

[16] Giovanni Cherubin, Rob Jansen, and Carmela Troncoso. 2022. Online Website Fingerprinting: Evaluating Website Fingerprinting Attacks on Tor in the Real World. In *Proceedings of the USENIX Security Symposium*. 753–770.

[17] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. 2015. Riposte: An anonymous messaging system handling millions of users. In *Proceedings of the IEEE Symposium on Security and Privacy*. 321–338.

[18] Henry Corrigan-Gibbs and Bryan Ford. 2010. Dissent: accountable anonymous group messaging. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 340–350.

[19] George Danezis. 2003. Statistical Disclosure Attacks. In *Security and Privacy in the Age of Uncertainty*. Springer US, 421–426.

[20] George Danezis and Carmela Troncoso. 2013. You cannot hide for long: De-anonymization of real-world dynamic behaviour. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society*. 49–60.

[21] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. 2018. Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency-choose two. In *Proceedings of the IEEE Symposium on Security and Privacy*. 108–126.

[22] Roger Dingledine and George Kadianakis. 2014. In *Proceedings of the 7th Workshop on Hot Topics in Privacy Enhancing Technologies*.

[23] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. Tor: The Second-Generation Onion Router. In *Proceedings of the USENIX Security Symposium*.

[24] Kevin Dyer, Scott Coull, Thomas Ristenpart, and Thomas Shrimpton. 2012. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *Proceedings of the IEEE Symposium on Security and Privacy*. 332–346.

[25] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. 2013. Protocol Misidentification Made Easy with Format-transforming Encryption. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 61–72.
[26] Matthew Edman and Paul Syverson. 2009. As-awareness in Tor Path Selection. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 380–389.
[27] Matthew Edman and Bülent Yener. 2009. On anonymity in an electronic society: A survey of anonymous communication systems. *ACM Computing Surveys (CSUR)* 42, 1 (2009), 1–35.
[28] Saba Eskandarian and Dan Boneh. 2022. Clarion: Anonymous communication from multiparty shuffling protocols. *Proceedings of the Network and Distributed System Security Symposium* (2022).
[29] Saba Eskandarian, Henry Corrigan-Gibbs, Matei Zaharia, and Dan Boneh. 2021. Express: Lowering the cost of metadata-hiding communication with cryptographic privacy. In *Proceedings of the USENIX Security Symposium*. 1775–1792.
[30] Srivatsava Ranjit Ganta, Shiva Prasad Kasiviswanathan, and Adam Smith. 2008. Composition attacks and auxiliary information in data privacy. In *Proceedings of the ACM SIGKDD international conference on Knowledge discovery and data mining*. 265–273.
[31] Sharad Goel, Mark Robson, Milo Polte, and Emin Sirer. 2003. *Herbivore: A scalable and efficient protocol for anonymous communication.* Technical Report. Cornell University.
[32] Oana Goga, Howard Lei, Sree Hari Krishnan Parthasarathi, Gerald Friedland, Robin Sommer, and Renata Teixeira. 2013. Exploiting innocuous activity for correlating users across sites. In *Proceedings of the international conference on World Wide Web*. 447–458.
[33] Jiajun Gong and Tao Wang. 2020. Zero-Delay Lightweight Defenses against Website Fingerprinting. In *Proceedings of the USENIX Security Symposium*. 717–734.
[34] Jiajun Gong, Wuqi Zhang, Charles Zhang, and Tao Wang. 2022. Surakav: Generating Realistic Traces for a Strong Website Fingerprinting Defense. In *Proceedings of the IEEE Symposium on Security and Privacy*. 1558–1573.
[35] Jamie Hayes, Carmela Troncoso, and George Danezis. 2016. TASP: Towards Anonymity Sets that Persist. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society*. 177–180.
[36] Nguyen Phong Hoang, Panagiotis Kintis, Manos Antonakakis, and Michalis Polychronakis. 2018. An empirical study of the I2P anonymity network and its censorship resistance. In *Proceedings of the Internet Measurement Conference*. 379–392.
[37] James K Holland and Nicholas Hopper. 2022. RegulaTor: A Straightforward Website Fingerprinting Defense. *Privacy Enhancing Technologies* 2 (2022).
[38] Nicholas Hopper and Eugene Y. Vasserman. 2006. On the Effectiveness of k-Anonymity against Traffic Analysis and Surveillance. In *Proceedings of the ACM Workshop on Privacy in Electronic Society*. 9–18.
[39] Peipei Jiang, Qian Wang, Jianhao Cheng, Cong Wang, Lei Xu, Xinyu Wang, Yihao Wu, Xiaoyuan Li, and Kui Ren. 2023. Boomerang: Metadata-Private Messaging under Hardware Trust. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*. 877–899.
[40] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. 2013. Users get routed: Traffic correlation on Tor by realistic adversaries. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 337–348.
[41] Seongmin Kim, Juhyeng Han, Jaehyeong Ha, Taesoo Kim, and Dongsu Han. 2017. Enhancing Security and Privacy of Tor's Ecosystem by Using Trusted Execution Environments. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation*. 145–161.
[42] Katharina Kohls, Kai Jansen, David Rupprecht, Thorsten Holz, and Christina Pöpper. 2019. On the Challenges of Geographical Avoidance for Tor. In *Proceedings of the Network and Distributed System Security Symposium*.
[43] Christiane Kuhn, Martin Beck, Stefan Schiffner, Eduard Jorswieck, and Thorsten Strufe. 2019. On Privacy Notions in Anonymous Communication. *Proceedings on Privacy Enhancing Technologies* 2019, 2 (2019), 105–125.
[44] Albert Kwon, Henry Corrigan-Gibbs, Srinivas Devadas, and Bryan Ford. 2017. Atom: Horizontally scaling strong anonymity. In *Proceedings of the Symposium on Operating Systems Principles*. 406–422.
[45] Albert Kwon, David Lu, and Srinivas Devadas. 2020. XRD: Scalable messaging system with cryptographic privacy. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*. 759–776.
[46] Albert Hyukjae Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. 2016. Riffle: An efficient communication system with strong anonymity. *Proceedings on Privacy Enhancing Technologies* 2016, 2 (2016), 115–134.
[47] David Lazar, Yossi Gilad, and Nickolai Zeldovich. 2018. Karaoke: Distributed private messaging immune to passive traffic analysis. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*. 711–725.
[48] David Lazar, Yossi Gilad, and Nickolai Zeldovich. 2019. Yodel: strong metadata security for voice calls. In *Proceedings of the ACM Symposium on Operating Systems Principles*. 211–224.
[49] David Lazar and Nickolai Zeldovich. 2016. Alpenhorn: Bootstrapping secure communication without leaking metadata. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*. 571–586.

[50] Stevens Le Blond, David Choffnes, William Caldwell, Peter Druschel, and Nicholas Merritt. 2015. Herd: A scalable, traffic analysis resistant anonymity network for VoIP systems. In *Proceedings of the ACM Conference on Special Interest Group on Data Communication*. 639–652.

[51] Stevens Le Blond, David Choffnes, Wenxuan Zhou, Peter Druschel, Hitesh Ballani, and Paul Francis. 2013. Towards efficient traffic-analysis resistant anonymity networks. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 303–314.

[52] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. 2007. t-closeness: Privacy beyond k-anonymity and l-diversity. In *Proceedings of the IEEE International Conference on Data Engineering*. 106–115.

[53] Zhihao Li, Stephen Herwig, and Dave Levin. 2017. Detor: Provably avoiding geographic regions in Tor. In *Proceedings of the USENIX Security Symposium*. 343–359.

[54] Anna Harbluk Lorimer, Lindsey Tulloch, Cecylia Bocovich, and Ian Goldberg. 2021. OUStralopithecus: Overt User Simulation for Censorship Circumvention. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society*. 137–150.

[55] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew Miller. 2019. Honeybadgermpc and asynchromix: Practical asynchronous MPC and its application to anonymous communication. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 887–903.

[56] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. 2007. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data* 1, 1 (2007).

[57] Nick Mathewson and Roger Dingledine. 2005. Practical Traffic Analysis: Extending and Resisting Statistical Disclosure. In *Privacy Enhancing Technologies*. Springer Berlin Heidelberg, 17–34.

[58] Aastha Mehta, Mohamed Alzayat, Roberta De Viti, Björn B Brandenburg, Peter Druschel, and Deepak Garg. 2022. Pacer: Comprehensive Network Side-Channel Mitigation in the Cloud. In *Proceedings of the USENIX Security Symposium*. 2819–2838.

[59] Roland Meier, Vincent Lenders, and Laurent Vanbever. 2022. Ditto: WAN traffic obfuscation at line rate. In *Proceedings of the Network and Distributed System Security Symposium*.

[60] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. 2018. DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 1962–1976.

[61] Milad Nasr, Amir Houmansadr, and Arya Mazumdar. 2017. Compressive traffic analysis: A new paradigm for scalable traffic analysis. In *Proceedings of the ACM Conference on Computer and Communications Security*. 2053–2069.

[62] Netflix. 2021. Netflix Internet Connection Speed Recommendations. https://help.netflix.com/en/node/306.

[63] Zachary Newman, Sacha Servan-Schreiber, and Srinivas Devadas. 2022. Spectrum: High-bandwidth Anonymous Broadcast. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*. 229–248.

[64] Rishab Nithyanand, Oleksii Starov, Adva Zair, Phillipa Gill, and Michael Schapira. 2016. Measuring and mitigating AS-level adversaries against Tor. In *Proceedings of the Network and Distributed System Security Symposium*.

[65] Vítor Nunes, José Brás, Afonso Carvalho, Diogo Barradas, Kevin Gallagher, and Nuno Santos. 2023. BriK source code. https://github.com/TheBriKProject/BriK.

[66] Se Eun Oh, Taiji Yang, Nate Mathews, James K Holland, Mohammad Saidur Rahman, Nicholas Hopper, and Matthew Wright. 2022. DeepCoFFEA: Improved Flow Correlation Attacks on Tor via Metric Learning and Amplification. In *Proceedings of the IEEE Symposium on Security and Privacy*. 1915–1932.

[67] Ania M Piotrowska. 2020. *Low-latency mix networks for anonymous communication*. Ph. D. Dissertation. UCL (University College London).

[68] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. 2017. The Loopix Anonymity System. In *Proceedings of the USENIX Security Symposium*. 1199–1216.

[69] Mohammad Saidur Rahman, Payap Sirinam, Nate Mathews, Kantha Girish Gangadhara, and Matthew Wright. 2020. Tik-Tok: The utility of packet timing in website fingerprinting attacks. *Proceedings on Privacy Enhancing Technologies* 2020, 3 (2020).

[70] Neil M Richards. 2013. The dangers of surveillance. *Harvard Law Review* 126, 7 (2013), 1934–1965.

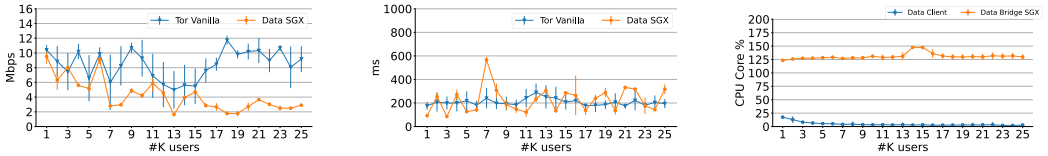
[71] Florentin Rochet, Ryan Wails, Aaron Johnson, Prateek Mittal, and Olivier Pereira. 2020. CLAPS: Client-Location-Aware Path Selection in Tor. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 17–34.

[72] Rafal Rohozinski Ronald Deibert, John Palfrey and Jonathan Zittrain (Eds.). 2010. *Access Controlled: The Shaping of Power, Rights, and Rule in Cyberspace*. MIT Press.

[73] Marc B Rosen, James Parker, and Alex J Malozemoff. 2021. Balboa: Bobbing and weaving around network censorship. In *Proceedings of the USENIX Security Symposium*. 3399–3413.

[74] Tianxiang Shen, Jianyu Jiang, Yunpeng Jiang, Xusheng Chen, Ji Qi, Shixiong Zhao, Fengwei Zhang, Xiapu Luo, and Heming Cui. 2021. DAENet: making strong anonymity scale in a fully decentralized network. *IEEE Transactions on Dependable and Secure Computing* 19, 4 (2021), 2286–2303.

[75] Fatemeh Shirazi, Milivoj Simeonovski, Muhammad Rizwan Asghar, Michael Backes, and Claudia Diaz. 2018. A survey on routing in anonymous communication protocols. *ACM Computing Surveys (CSUR)* 51, 3 (2018), 1–39.
[76] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, and Prateek Mittal. 2015. RAPTOR: Routing Attacks on Privacy in Tor. In *Proceedings of the USENIX Security Symposium*. 271–286.
[77] Zhen Sun and Vitaly Shmatikov. 2023. Telepath: A Minecraft-based Covert Communication System. In *Proceedings of the IEEE Symposium on Security and Privacy*. 2223–2237.
[78] Tor Project. [n. d.]. Pluggable Transport Specification. https://gitweb.torproject.org/torspec.git/tree/pt-spec.txt.
[79] Tor Project. 2014. Traffic correlation using netflows. https://blog.torproject.org/traffic-correlation-using-netflows?page=1.
[80] Tor Project. 2018. Reporting Bad Relays. https://trac.torproject.org/projects/tor/wiki/doc/ReportingBadRelays.
[81] Tor Project. 2019. meek. https://trac.torproject.org/projects/tor/wiki/doc/meek.
[82] Tor Project. 2019. Tor FAQ. https://2019.www.torproject.org/about/overview.html.en.
[83] Tor Project. 2019. Tor Pluggable Transports. https://2019.www.torproject.org/docs/pluggable-transports.
[84] Tor Project. 2020. Tor control protocol. https://gitweb.torproject.org/torspec.git/tree/control-spec.txt.
[85] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. 2017. Stadium: A distributed metadata-private messaging system. In *Proceedings of the Symposium on Operating Systems Principles*. 423–440.
[86] Adithya Vadapalli, Kyle Storrier, and Ryan Henry. 2022. Sabre: Sender-Anonymous Messaging with Fast Audits. In *Proceedings of the IEEE Symposium on Security and Privacy*. 1953–1970.
[87] Jelle Van Den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. 2015. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the Symposium on Operating Systems Principles*. 137–152.
[88] Ryan Wails, Yixin Sun, Aaron Johnson, Mung Chiang, and Prateek Mittal. 2018. Tempest: Temporal Dynamics in Anonymity Systems. *Proceedings on Privacy Enhancing Technologies* 2018, 3 (2018), 22–42.
[89] Gerry Wan, Aaron Johnson, Ryan Wails, Sameer Wagh, and Prateek Mittal. 2019. Guard placement attacks on path selection algorithms for Tor. *Proceedings on Privacy Enhancing Technologies* 2019, 4 (2019).
[90] Lauren Watson, Anupam Mediratta, Mohammad Tariq Elahi, and Rik Sarkar. 2020. Privacy Preserving Detection of Path Bias Attacks in Tor. *Proceedings on Privacy Enhancing Technologies* 2020, 4 (2020), 111–130.
[91] Philipp Winter, Richard Köwer, Martin Mulazzani, Markus Huber, Sebastian Schrittwieser, Stefan Lindskog, and Edgar Weippl. 2014. Spoiled onions: Exposing malicious Tor exit relays. *Proceedings on Privacy Enhancing Technologies* (2014).
[92] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. 2012. Dissent in numbers: Making strong anonymity scale. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*. 179–182.
[93] David Isaac Wolinsky, Ewa Syta, and Bryan Ford. 2013. Hang With Your Buddies to Resist Intersection Attacks. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 1153–1166.
[94] Charles V. Wright, Scott E. Coull, and Fabian Monrose. 2009. Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis. In *Proceedings of the Network and Distributed System Security Symposium*.



(a) Throughput. (b) Latency. (c) CPU consumption.

Fig. 9. Throughput, latency, and CPU consumption of BriK (`FSize` = 3125 B, `TSRate` = 5 Mbps) within SGX.

A THWARTING BRIDGE HIJACKING ATTACKS

To maintain the anonymity properties of k-funnels, BriK bridges must be trusted (1) to *not reveal* the identity of participating clients and (2) to *follow the protocol* between hubs and bridge. However, powerful adversaries may deploy malicious bridges or compromise benign ones, allowing them to obtain sensitive information and potentially deanonymize clients. For instance, access to a hub's outbound queues would trivially allow the adversary to identify all the Tor circuit cells relayed by that hub. Thus, it is essential that the execution state of a BriK bridge preserves two main security properties: confidentiality and integrity.

A.1 Securing BriK Bridges Inside a TEE

To detect potentially untrusted bridges, we propose securing the BriK bridge software inside a trusted execution environment (TEE), enabled by trusted hardware. Inspired by the approach of Kim et al. [41], our solution requires that the bridge server is equipped with trusted TEE-enabling hardware. The TEE provides a protected domain where the bridge program's execution state is isolated from the host's OS, guaranteeing confidentiality and integrity. Remote attestation executed at the hub can verify that the bridge is authentic before joining any k-funnels. In our implementation, we leveraged Intel SGX to protect the bridge software inside an enclave. We used SCONE [7] which allows Docker containers to use Intel SGX. Supporting SGX required no significant changes in our code. Since SCONE provides a standard C library interface, we created a Docker container and recompiled BriK using a C++ cross-compiler with SCONE support.

A.2 Performance of SGX-enabled BriK Bridges

By leveraging the memory isolation capabilities provided by SGX, BriK can protect data structures that store sensitive information about the members of each k-funnel and BriK reception queues. We repeated the performance measurements presented in §5.2 using an SGX-enabled machine equipped with a common-of-the-shelf Intel Core i9-9900K CPU. Overall, our results suggest that SGX-enabled BriK bridges could be easily adopted in practice.

Figure 9 a) depicts the throughput obtained by BriK clients connected to an SGX-enabled bridge. We can see that, similarly to our observations when using a non-SGX BriK bridge, throughput is reduced as new clients join the k-funnel. In particular, while clients can achieve a throughput of about 8Mbps in a k-funnel composed of just three users, the throughput achieved by a k-funnel with $k \geq 18$ sits between 2 and 4Mbps. In turn, Figure 9 b) shows that the latency penalty experienced by clients when using the SGX-infused version of BriK is relatively close to that of vanilla Tor, for small and large numbers of k.

Regarding CPU consumption, we can also observe that an SGX-enabled bridge is able to serve up to 25 BriK clients while consuming only about 1.25 CPU cores of the 2-core Docker container living in the host machine. The fact that CPU consumption remains relatively stable throughout our experiment may be explained by the fact that, for this particular experiment, we had to compile BriK statically to ensure compatibility with SCONE (BriK's non-SGX version was compiled dynamically).

B BRIK MESSAGE FORMATS

Hubs and bridges exchange three different frame types: data, control, and chaff. Each frame, of fixed size, has a header and a payload. The first field of a frame's header carries the type, and additional header fields are dependent on it. Next, we describe the composition of each frame type.

- *Data frames:* Data frames include one additional header field, the payload size, which denotes the length of useful bytes carried in the frame. Whenever Tor sends a cell to BriK, the cell is enclosed in one or several data frames, which are later sent. Our frame format, depicted in Figure 10 allows BriK frames to be large enough to contain multiple Tor cells, without fragmenting those Tor cells across several frames. Whenever the payload size is smaller than the fixed frame size, padding is appended to the payload.
- *Control frames:* Control frames always include one more header field: the control type. Depending on the control type, several additional fields may be included as depicted in Figure 11. These frames are used for managing k-funnels. `Hello` commands allow users to define their k-min to join a k-funnel. `TS Rate` commands allow bridges to dynamically modify the hub frame sending rate. Like data frames, padding is added to control frames filling the frame up to its size. Control frames have the highest priority of the three types.
- *Chaff frames:* Chaff frames do not include any additional header fields and their payload consists entirely of arbitrary bytes. When no Tor traffic is available, BriK sends chaff frames. Such frames are discarded upon reception both at the bridge and hubs. Chaff frames have the lowest priority of the three types.

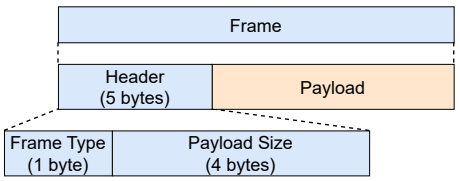


Fig. 10. Format of BriK data frames.

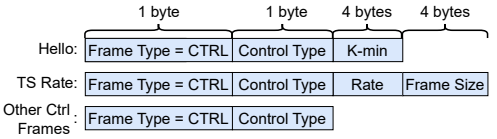


Fig. 11. Format of BriK control frame headers.

C ALTERNATIVE TRAFFIC SHAPING PATTERNS

The generation of chaff traffic based on the transmission of constant traffic that is independent of the underlying application's actual workload (as we do in BriK), can be inefficient. To tackle this issue, recent works like Ditto [59] and Pacer [58] propose profiling routines that compute and suggest more efficient chaff traffic shaping patterns according to the specific tasks users are expected to perform more often (e.g., video streaming, e-mail, web browsing, etc.), while ensuring that these traffic shaping patterns do not leak information to network adversaries. An interesting direction for future work involves exploring the application of efficient traffic shaping techniques within BriK's traffic shaper to enhance QoS while preventing adversaries from deducing users' real traffic patterns, e.g., by having organizations profile user traffic before deploying BriK.

D DEFENDING AGAINST LONG TERM STATISTICAL DISCLOSURE

To protect against statistical disclosure in the long run, we propose an approach inspired by Wolinsky et al. [93], initially conceived for message-based anonymity networks.

Specifically, a BriK bridge defines a round as a period of time, δ, and contains two components: an oracle function O and a policy P. For reach round r, Alice can send any number of messages. After δ seconds pass, r increments. O then performs a statistical disclosure attack using r and the messages observations $\bar{O}$ up until round r [57] using circuits as proxies for destination servers. This

Device	ISP	Modem	Wired	Avg. Tput. (Mbps)	Avg. Latency (ms)
rpi01	ISP_1	Fiber	WiFi	**85.51 ± 2.48**	**46.24 ± 1.77**
rpi02	ISP_1	Fiber	Ethernet	**194.96 ± 2.43**	**44.69 ± 0.30**
rpi03	ISP_2	Fiber	Ethernet	**203.43 ± 3.42**	**41.91 ± 1.35**
rpi04	ISP_3	Coaxial	Ethernet	**17.13 ± 2.35**	**49.18 ± 7.26**
rpi05	ISP_3	Coaxial	Ethernet	**21.38 ± 0.82**	**52.07 ± 25.70**
rpi06	ISP_2	Fiber	Ethernet	**117.90 ± 0.84**	**49.01 ± 3.37**
rpi07	ISP_2	Fiber	Ethernet	**92.92 ± 1.94**	**57.58 ± 165.15**
rpi08	ISP_1	Fiber	WiFi	**92.71 ± 0.78**	**118.94 ± 339.71**
rpi09	ISP_2	Fiber	Ethernet	**119.01 ± 0.48**	**42.49 ± 4.91**

Table 1. Connectivity characteristics and network performance of the deployed RPi BriK nodes.

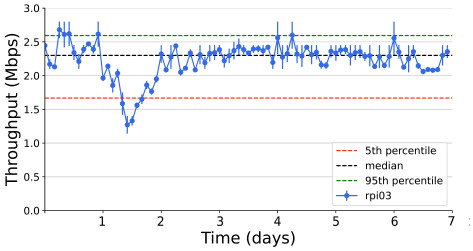


Fig. 12. BriK throughput over six days (*rpi3*).

results in $\vec{v} = O(r, \bar{O})$. The oracle then queries P for the current policy and receives (c, e) where c is a specified level of confidence provided by the policy and e is a margin of error. O then can look at the resulting probability distribution, $\vec{v}$, and look at all circuits in $\vec{v}$ with probability difference less than or equal to c, resulting in set C. Then, O looks at the set A of circuits owned by Alice to see if any circuits in C are correctly predicted.

If $C \cap A \neq \emptyset$, the oracle considers the attack successful and the BriK bridge notifies the client that it should enter a chaff-only mode. Alice then switches to sending only chaff messages to the BriK bridge. During this time O continues performing the statistical disclosure attack as described before. However, since Alice is in chaff-only mode, O will look at all circuits in $\vec{v}$ where probability difference is lower than $c + e$, where e is an error parameter specified by the policy. When O computes C s.t. $C \cap A = \emptyset$, the BriK bridge will notify Alice that she can leave chaff-only mode.

This conservative estimate allows a BriK bridge to determine if a statistical analysis attack would work while circuits are alive, but it does nothing to determine the long-term effect of churn on BriK bridges. For this, whenever churn occurs on a BriK bridge, all clients $u \in U$ may come together and leverage a multiparty computation (MPC) protocol to create a distributed oracle O using an agreed upon policy P. Each client's private input includes their message history $m_u \forall u \in U$. The output of O will be a set of users $V \subseteq U$ who are estimated to be at heightened risk of statistical disclosure attack. Those clients may choose to enter chaff-only mode to lower the success rate of such an attack. We leave implementing this MPC oracle O to future work.

E RAW NETWORK PERFORMANCE

Table 1 details the connectivity characteristics of each RPi machine we deployed as BriK hubs on the metropolitan area of a major European city (see §5.1). The table also shows the average throughput and latency obtained by each node over the course of a six days-long measurement where throughput and latency statistics were obtained every two hours. These statistics were obtained by reaching out to a public server under our control, hosted within a different European country, using the `httping` and the `iperf3` utilities, respectively. We did not send traffic through Tor or BriK during these measurements, i.e., we intended to measure the raw network performance.

Figure 12 shows our throughput measurements conducted over BriK. The figure reveals that BriK's traffic was rather stable, achieving a median throughput of 2.3 Mbps. Apart from the performance variability of Tor circuits, which inherently affect BriK's end-to-end performance, we were able to observe some dips in throughput (e.g., day 1–2). This can be explained by the fact that the home network of *rpi3*'s operator was significantly overloaded during this period.

F THROUGHPUT IN A CLOUD ENVIRONMENT

Figure 13 depicts the results of a throughput measurement experiment we conducted when the BriK bridge is deployed in a cloud machine on a Google datacenter in Europe and all hubs are

spread out across three different continents: America, Europe, and Asia (see §5.1). The plot reveals that well-connected hubs are able to achieve an overall higher throughput than when compared to hubs connected to residential networks (Figure 12). However, we found throughput oscillations to be more pronounced on the cloud deployment, likely due to the distributed nature of nodes across the world, which might be impacted by global WAN connectivity.

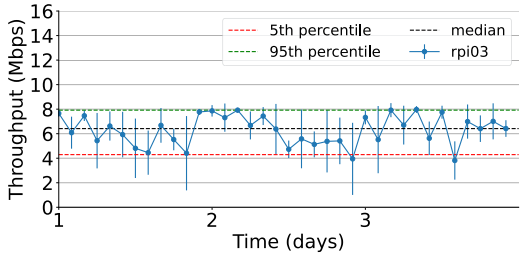


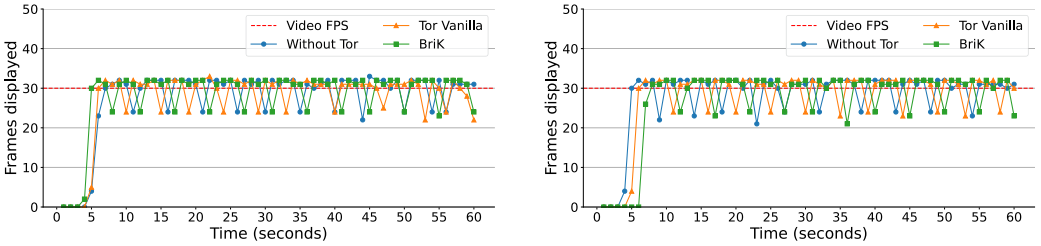
Fig. 13. BriK throughput over the course of three days (*rpi3*) when the other 9 hubs are deployed in the cloud.

G VIDEO STREAMING

To assess different QoS indicators of BriK besides raw throughput, we explored the quality of video streaming activities over k-funnels when our prototype is configured to operate with different traffic shaping rates (see §5.3).

Figure 14(a) shows the number of frames displayed to BriK over the span of one minute, when streaming a 480p resolution video. We can see that the number of frames received and displayed by the client's video software (VLC) is rather consistent when leveraging BriK– i.e., regularly reported to be close to 30 FPS, with some occasional drops to 23 FPS. Figure 14(b) reveals a very similar behavior for 720p resolution video, suggesting that BriK can accommodate for more demanding workloads by changing its traffic shaping rate.

The figures also show that BriK does not introduce a meaningful quality of service degradation vs. when the same tasks as performed through vanilla Tor. As can be observed in both figures, a client leveraging vanilla Tor to perform the video streaming workload also regularly reports 30 FPS with occasional FPS drops throughout our experiment.



(a) Streaming a 480p video.

(b) Streaming a 720p video.

Fig. 14. Number of frames shown to BriK's client over a 60 seconds-long HTTP 30 FPS video streaming.

Received July 2023; revised September 2023; accepted October 2023