Ludovic Barman*, Italo Dacosta,
Mahdi Zamani, Ennan Zhai, Apostolos Pyrgelis, Bryan Ford, Joan Feigenbaum, and Jean-Pierre Hubaux

PriFi: Low-Latency Anonymity for Organizational Networks

Abstract: Organizational networks are vulnerable to traffic-analysis attacks that enable adversaries to infer sensitive information from network traffic — even if encryption is used. Typical anonymous communication networks are tailored to the Internet and are poorly suited for organizational networks. We present PriFi, an anonymous communication protocol for LANs, which protects users against eavesdroppers and provides high-performance traffic-analysis resistance. PriFi builds on Dining Cryptographers networks (DC-nets), but reduces the high communication latency of prior designs via a new client/relay/server architecture, in which a client's packets remain on their usual network path without additional hops, and in which a set of remote servers assist the anonymization process without adding latency. PriFi also solves the challenge of equivocation attacks, which are not addressed by related work, by encrypting traffic based on communication history. Our evaluation shows that PriFi introduces modest latency overhead (≈100ms for 100 clients) and is compatible with delay-sensitive applications such as Voice-over-IP.

Keywords: anonymity, DC-nets, traffic analysis, local-area networks, communications

DOI 10.2478/popets-2020-0061
Received 2020-02-29; revised 2020-06-15; accepted 2020-06-16.

1 Introduction

Local-area networks (LANs and WLANs) deployed in organizational networks are vulnerable to eavesdropping attacks. Sensitive traffic is usually encrypted, but metadata such as *who is communicating* and the communication patterns remain visible. Such metadata enable an eavesdropper to identify and track users passively [30, 87], to infer contents and endpoints [8, 9, 27, 36, 56, 68, 79, 81], and potentially to perform targeted attacks on high-value devices and users. Eavesdropping attacks can be performed by a single compromised endpoint or malicious user. This is particularly worrisome when the users are loosely trusted, or when the organizational network is deployed in an adverse environment. For example, the International Committee of the Red Cross (ICRC) has strong privacy and security needs regarding their communications: a previous study confirms that its "staff and beneficiaries need to communicate in a multitude of adverse environments that are often susceptible to eavesdropping, to physical attacks on the infrastructure, and to coercion of the personnel" [51].

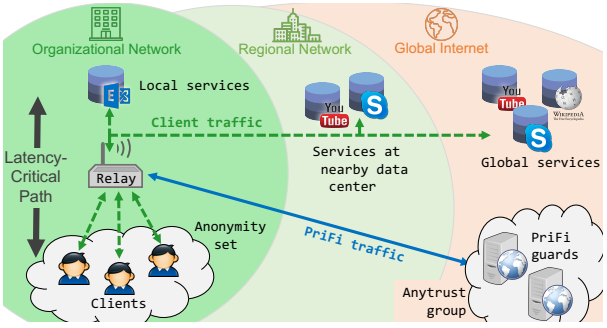


Fig. 1. PriFi's architecture consisting of clients, a relay, and a group of anytrust servers called the *guards*. Clients' packets remain on their usual network path without additional hops, unlike in mix-networks and onion-routing protocols.

To protect against eavesdropping in LANs, few solutions exist today. Anonymous communication networks (ACNs) are designed to conceal the communicating entities; however, most ACNs are designed for the Internet and translate poorly to the LAN setting. Most ACNs rely on mix-networks or onion-routing, a common drawback of which is that their security relies on routing the traffic through a series of servers distributed around the Internet [11, 12, 26, 42, 60]. First, this implies that internal communications in the organization's network would need to be routed over the Internet. More importantly, to minimize the risks of coercion and collusion, these servers are typically spread across different jurisdictions, hence these designs introduce significant latency overhead.

***Corresponding Author: Ludovic Barman:** EPFL, E-mail: ludovic.barman@epfl.ch
Italo Dacosta: UBS, E-mail: italo.dacosta@epfl.ch
Mahdi Zamani: Visa Research, E-mail: mzamani@visa.com
Ennan Zhai: Alibaba Group, E-mail: ennan.zhai@alibaba-inc.com
Apostolos Pyrgelis: EPFL, E-mail: apostolos.pyrgelis@epfl.ch
Bryan Ford: EPFL, E-mail: bryan.ford@epfl.ch
Joan Feigenbaum: Yale University, E-mail: joan.feigenbaum@yale.edu
Jean-Pierre Hubaux: EPFL, E-mail: jean-pierre.hubaux@epfl.ch

Dining Cryptographers networks (DC-nets) [10] are an anonymization primitive that could be attractive in terms of latency in some contexts, as their security relies on information coding and not on sequential operations done by different servers. In theory, therefore, anonymity can be achieved without high-latency server-to-server communication. This theoretical appeal has not been achieved in practice, however. Previous DC-net systems such as Dissent [14], Dissent in Numbers [83], and Verdict [15] still use costly server-to-server communication, thus imposing latencies in the order of seconds [83], and notably routing users' traffic through all of the servers.

We present PriFi, the first low-latency anonymous communication network tailored to organizational networks. PriFi provides anonymity against global eavesdroppers: Users are assured that their communication patterns are indistinguishable from the communications of other PriFi users, even if the local network infrastructure is compromised. To anonymize IP packet flows, PriFi works at the network level like a VPN. Unlike a VPN, however, PriFi's security does not depend on a single endpoint, and the protocol provably resists traffic analysis. PriFi provides low-latency, traffic-agnostic communication suitable for delay-sensitive applications such as streaming and VoIP, at the cost of higher LAN bandwidth usage.

Compared with previous work, PriFi significantly reduces communication latency through a new three-tier architecture composed of *clients*, a *relay* in the LAN (*e.g.,* a router), and *guard* servers that are geographically distributed over the Internet (Figure 1). This architecture is compatible with organizational networks, and enables PriFi to avoid major latency overheads present in other ACNs. Unlike previous DC-net systems that use multi-hop, multi-round protocols, and costly server-to-server communications [14, 15, 83], PriFi achieves similar guarantees while removing all server-to-server communications from the latency-critical path. To produce anonymous output, PriFi ciphertexts pre-computed and sent by the guards are combined locally at the relay, so that relay↔guard delay does not affect the latency experienced by clients. Moreover, the traffic from clients remains on its usual network path, client↔relay↔destination, and does not go through the guards. Some added latency results from buffering and software processing, but not from additional network hops. As a result, PriFi's latency is 2 orders of magnitude lower than the closest related work with the same setup [83].

We also present a solution for *equivocation attacks*, *i.e.,* de-anonymization by a malicious relay sending different information to different clients and analyzing their subsequent behavior. Previous DC-net systems are vulnerable to this attack, but do not address it [14, 15, 83]. Equivocation attacks can be detected using consensus or gossiping between the clients, at a high bandwidth and latency cost. We present a new low-latency, low-bandwidth solution that relies on binding encryption to communication history.

We evaluate PriFi on a topology corresponding to an organizational network. We observe that the latency overhead caused by PriFi is low enough for VoIP and video conferencing (≈100 ms for 100 users), and that the internal and external bandwidth usage of PriFi is acceptable in an organizational network (≈40 Mbps in a 100 Mbps LAN). In comparison, the latency of the closest related work, Dissent in Numbers [83], is 14.5 seconds for 100 clients on the same setup. One part of the evaluation is dedicated to the ICRC scenario; we replay real network traces recorded at an ICRC delegation and find that the increase in latency is tolerable in practice (between 20 and 140ms on average).

In this paper, we make the following contributions:

- PriFi, a low-latency, traffic-agnostic, traffic-analysis-resistant anonymous communication network, building on a new DC-nets architecture optimized for LANs;
- A low-latency method of protecting DC-nets against disruption attacks (*i.e.,* jamming) by malicious insiders;
- A new low-latency defense against equivocation attacks;
- An open-source implementation of PriFi, tested and evaluated on desktop computers, and implementations for Android and iOS [62];
- An analysis of the effect of user mobility on DC-nets.

2 Background on DC-nets

A Dining Cryptographers network or DC-net [10] is a protocol that provides anonymous broadcast for a group of users who communicate in lock-step, in successive rounds. In a given round, each user produces a ciphertext of the same length. One user also embeds a plaintext in its ciphertext. Combining all users' ciphertexts reveals the plaintext, without revealing *which* user sent it.

Figure 2 shows an example of a 1-bit DC-net. Each pair of users derives a shared secret (in red). Each user's ciphertext is the XOR of his shared secrets (in green). Bob, the anonymous sender, also XORs in its message (in blue). By XORing all ciphertexts together, all shared secrets cancel out, revealing the anonymized message. If at least two users are honest, this protocol achieves unconditional sender anonymity. Every user sends a ciphertext of the same length, ciphertexts from honest parties are indistinguishable from each other without all

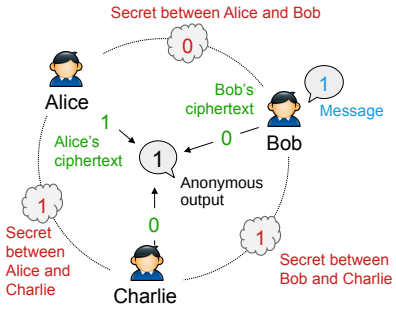


Fig. 2. Example of a 1-bit DC-net.

shared secrets, and a single missing ciphertext (from an honest party) prevents the computation of the output.

In practice, to produce ciphertexts for multiple rounds, shared secrets are used to seed pseudo-random generators.

Impact of Topology. The original DC-net design [10] requires key material between every pair of members. Dissent [83] and subsequent work [15] present a more scalable two-tier topology made of clients and servers, which reduces the number of keys. However, the client/server topology has a significant negative impact on latency: It requires several server-to-server rounds of communication to ensure integrity, accountability, and to handle churn. PriFi avoids this drawback with a new client/relay/guard architecture.

Disruption Protection. Vanilla DC-nets are vulnerable to disruption attacks from malicious insiders [10], where a malicious user can corrupt other clients' messages. Some previous work uses proactively verifiable constructions that are too slow for low-latency communication [15]. Others use "trap bits" and "blame mechanisms" [84] that require minutes to hours to find disruptors, mainly due to expensive server-to-server communication. PriFi uses a new retroactive blame mechanism to detect disruptors in seconds.

Equivocation Protection. Answers to anonymous messages are typically broadcast to all users. Previous DC-net designs did not address equivocation attacks, where a malicious server sends each client different, identifiable information to distinguish the anonymous receiver (see Section 6). Equivocation thus represents a practical and covert attack vector against prior systems [14, 15, 83]. Equivocation attacks can be detected using consensus [44] or gossiping [70] between the clients, at a high bandwidth and latency cost. In PriFi, in contrast, messages from clients cannot be decrypted if an equivocation attack occurs, and PriFi protects against this threat without communication among clients.

3 System Overview

PriFi is similar to a low-latency relay or gateway service within a LAN, like a VPN or SOCKS proxy, which tunnels traffic between clients and the relay (*e.g.,* a LAN router). Informally, these tunnels protect honest clients' traffic from eavesdropping attacks. The traffic is anonymized, preventing a third-party from assigning a packet or flow to a specific device or end-user. Additionally, unlike traditional proxy services, (1) the relay need not be trusted, *i.e.,* security properties hold in case of compromise, and (2) the communications provably resist traffic-analysis attacks.

3.1 System Model

Consider n clients $C_1, \ldots, C_n$ that are part of an organizational network and are connected to a *relay* R. The relay is the gateway that connects the LAN to the Internet (*e.g.,* a LAN or WLAN router, Figure 1) and typically is already part of the existing infrastructure. The relay can process regular network traffic, in addition to running the PriFi software. Hence, PriFi can be deployed onto an existing network with minimal changes.

In the Internet, there is a small set $S_1, \ldots, S_m$ of m servers, called *guards*, whose role is to assist the relay in the anonymization process. These guards could be maintained by independent third parties, similar to Tor's volunteer relays, or sold as a "privacy service" by companies. To maximize diversity and collective trustworthiness, these guards are distributed around the world, preferably across different jurisdictions. Therefore, the connections between the guards and the relay are assumed to be high latency.

3.2 Threat Model

Let $\mathcal{A}$ be a computationally-bounded global passive adversary who observes all network traffic. In addition to the passive adversary, as PriFi is a closed-membership system, we consider and address active attacks from insiders, but not active attacks from outsiders, which are fairly orthogonal to PriFi and can be addressed via adequate server provisioning [61] or denial-of-service protection [57].

Most clients may be controlled by the adversary $\mathcal{A}$, but we require at least two honest clients at all times: otherwise, de-anonymization is trivial. The guards are in the *anytrust* model [14, 77, 83]: we assume that least one guard is honest, but a client does not need to know which one, and we assume all guards are highly available.

The relay is considered *malicious but available*: it may actively try to de-anonymize honest users or perform arbitrary attacks, but it will not perform actions that only affect the availability of PriFi communications such as delaying, corrupting, or dropping messages. On one hand, a fully-malicious relay would make little sense in our setting, where it is the gateway that connects the LAN to the Internet: due to its position in the network, it can degrade or deny service for any protocol anyway (*e.g.,* drop all packets). However, the relay is part of the infrastructure of the organization, and users would take administrative actions if the network is not operating properly. On the other hand, an adversarial model where the relay is honest-but-curious would be too weak: In practice, if the relay is compromised (*e.g.,* it gets hacked), it could perform active attacks to de-anonymize clients. Therefore, we use the "malicious but available" formulation to reflect that the relay needs to forward messages faithfully in order to provide service, but if the relay maliciously attacks the protocol, *only* availability and not user privacy will suffer.

3.3 Goals

3.3.1 Security Goals

- **[G1] Anonymity**: An adversary has a negligible advantage in attributing an honest user's message to its author. This includes traffic-analysis resistance: *e.g.,* the adversary can observe network-level traffic features.
- **[G2] Accountability**: Misbehaving insiders are traceable without affecting the anonymity of honest users.[1]

3.3.2 System Goals

- **[G3] Low latency**: The delay introduced by PriFi should be small enough to support network applications with high QoS requirements, *e.g.,* Voice-over-IP (VoIP) and videoconferencing applications.
- **[G4] Scalability**: One PriFi relay should support small to medium organizations of up to a few hundred users, a number typically observed in ICRC sites (Figure B.4).

3.3.3 Non-Goals

PriFi does not target the following goals:

- Hiding *all* traffic features: PriFi protects an honest user's traffic among all honest users' traffic, but does not hide global/aggregate communication volumes or time series of packets. Informally, an eavesdropper could learn that *some* honest user is browsing the Web or using VoIP, but not *which* honest user. Yet, this point is fairly orthogonal to the design of PriFi and can be addressed by adding padding and/or dummy traffic, at the cost of higher bandwidth usage, as proposed by substantial related work [29, 52, 80, 82, 86].
- External sender/receiver anonymity: PriFi's anonymity set consists of the LAN users connected to a relay. Users outside the LAN are not anonymous. If both sender and receiver are part of a PriFi LAN, not necessarily the same, the protocol has sender and receiver anonymity.
- Intersection attacks, which correlate users' presence on the PriFi network with messages or other users, are a practical threat to almost all ACNs [18, 85]. Although PriFi has no perfect solution to this problem, we discuss mitigation in Section 8, after presenting the system.

3.4 PriFi Solution Overview

PriFi starts with a setup phase where clients authenticate themselves to the relay. Clients and guards then derive shared secrets. Finally, clients are organized in a *schedule* (a secret permutation) to decide when they communicate.

Upstream Traffic. The clients and the guards run a DC-net protocol. Communication occurs in short *time slots*. In each time slot, each client and guard sends a ciphertext to the relay. The *slot owner* can additionally embed some payload. The relay waits for all ciphertexts, then computes the anonymized output. This reveals one or more IP packet(s) without source address; the relay replaces it with its own IP address (as in a NAT) and forwards it to its destination.

Due to the construction of the DC-net, this protocol ensures provable anonymity. Ciphertexts are indistinguishable from each other to the adversary. During a slot, each client sends exactly the same number of bits. Finally, if the contribution from any honest client is missing, the output is undecipherable. Informally, this property achieves our goal G1 (Section 3.3) for upstream traffic.

Precomputation of Ciphertexts. The ciphertexts from the guards are independent of the anonymous payloads. Hence, a key optimization is that guards' ciphertexts are batch computed and sent in advance to the relay. The relay buffers and pre-combines the ciphertexts from multiple guards, storing a single stream of pseudo-random bits to be later combined with clients' ciphertexts. This enables

1 This definition of accountability should not be confused with other definitions in which participants may de-anonymized based on communication content, *i.e.,* if someone does not like what they say.

PriFi to have low latency despite the presence of high-latency links between the guards and the relay.

Latency-Critical Path. Another important advantage of combining locally the ciphertexts is that clients' packets remain on their usual network path. The added latency is due mostly to the relay's need to wait for all clients. Similar systems that route clients' traffic between servers distributed around the Internet incur much higher latency.

Downstream Traffic. When receiving an answer to an anonymous message sent in some time slot, the relay encrypts it under the (anonymous) slot owner's public key, then broadcasts the ciphertext to all clients. As each client receives exactly the same message, this achieves our goal G1 (Section 3.3) for downstream traffic.

For broadcasting the downstream message, rather than performing n unicast transmissions, the relay exploits the LAN topology and uses UDP broadcast, letting layer-2 network equipment (*e.g.,* switches) replicate the message if needed. In WLANs, such a broadcast requires only one message, achieving receiver anonymity at no bandwidth or energy cost in the absence of link-layer retransmissions.

4 Basic PriFi Protocol

4.1 Preliminaries

Let λ be a standard security parameter, and let $\mathbb{G}$ be a cyclic finite group of prime order where the Decisional Diffie-Hellman (DDH) assumption [5] holds (*e.g.,* an elliptic curve).

Let $(\texttt{KeyGen}, \mathcal{S}, \mathcal{V})$ be a signature scheme, with $\texttt{KeyGen}(\mathbb{G}, 1^\lambda)$ an algorithm that generates the private-public key pair (p,P) used for signing. We denote as $\mathcal{S}_p(m)$ the signature of the message m with the key p.

Let $\texttt{KDF}: \mathbb{G}(1^\lambda) \to \{0,1\}^\lambda$ be a key derivation function that converts a group element into a bit string that can be used as a symmetric key. Let $(\mathcal{E}, \mathcal{D})$ be a symmetric nonce-based encryption scheme [65]. We denote as $\mathcal{E}_k(m)$ the encryption of the message m with the key k.

Let $H: \{0,1\}^* \to \{0,1\}^\lambda$ be a standard cryptographic hash function. Let $\texttt{PRG}: \{0,1\}^\lambda \to \{0,1\}^*$ be a standard pseudo-random generator. Let $F_1: \{0,1\}^\lambda \to \mathbb{G}$ be a public, invertible mapping function from binary strings to $\mathbb{G}$, and let $F_2: \{0,1\}^* \to \mathbb{G}$ be a hash function that maps bitstrings of arbitrary length to any point in $\mathbb{G}$ with uniform probability (*e.g.,* Elligator Squared [74]). Finally, let $F_3: \{0,1\}^* \to \mathbb{N}$ be a public function that maps bitstrings to integers.

Identities. Each party has a long-term key pair (denoted with the *hat* symbol) generated with $\texttt{KeyGen}(\mathbb{G}, 1^\lambda)$:

- $(\hat{p}_i, \hat{P}_i)$ for client C_i, with $i \in \{1,...,n\}$
- $(\hat{p}_j, \hat{P}_j)$ for guard S_j, with $j \in \{1,...,m\}$
- $(\hat{p}_r, \hat{P}_r)$ for the relay

Let $\underline{v}$ be the vector notation for v. For each epoch, the group definition G consists of all long-term public keys $G = (\underline{\hat{P}_i}, \underline{\hat{P}_j}, \hat{P}_r)$, $i \in \{1,...,n\}$, $j \in \{1,...,m\}$, and G is known to all parties (*e.g.,* via a public-key infrastructure). Finally, let $T = (\hat{P}_1, \hat{P}_2, ...)$ be a static roster of allowed clients known to the relay and the clients (*e.g.,* via a configuration file).

4.2 Protocols

PriFi starts with the protocol `Setup` (Protocol 1), followed by several runs of the protocol `Anonymize` (Protocol 2).

4.2.1 Setup

Each client authenticates itself to the relay using its long-term public key, and generates a fresh ephemeral key-pair. Then, each client C_i runs an authenticated Diffie-Hellman key exchange protocol with each guard S_j, using the fresh key pair to agree on a shared secret r_{ij}. This secret is used later to compute the DC-net's ciphertexts.

Then, to produce a permutation π, the guards shuffle the client's ephemeral public keys $\underline{P_i}$ by using a verifiable shuffle (*e.g.,* Neff's verifiable shuffle [54]). The public keys in π correspond to the keys in $\underline{P_i}$, in a shuffled order, such that no one knows the full permutation. Only a client holding the private key p_i corresponding to an input in $\underline{P_i}$ can recognize the corresponding pseudonym key in π.

`Setup` has the following properties (proved in A.1):

Property 1. A shared secret r_{ij} between an honest client C_i and an honest guard S_j is known only to C_i and S_j.

Property 2. Let C_0 and C_1 be two honest clients who ran `Setup` without aborting, and $\alpha(0), \alpha(1)$ the position of their respective shuffled key in π. Then, the adversary $\mathcal{A}$ has negligible advantage in guessing $b \in [0,1]$ such that the mapping client $\to$ position $(b \to \alpha(0), (1-b) \to \alpha(1))$ is in π.

Remark. The setup protocol (client/server secret sharing with a verifiable shuffle of client pseudonyms) is similar to that used in closely-related work [14, 83].

Protocol 1: `Setup`

Inputs: $\lambda, \mathbb{G}, G, T$

Outputs: schedule π, shared secrets r_{ij} between each client/guard pair (C_i, S_j)

1. Client→Relay Auth. Each client C_i generates a fresh key pair $(p_i, P_i) \leftarrow \texttt{KeyGen}(\mathbb{G}, 1^\lambda)$ and sends $P_i, \mathcal{S}_{\hat{p}_i}(P_i)$ to the relay. The relay checks the signature and that $\hat{P}_i \in T$, and it replies with $\mathcal{S}_{\hat{p}_r}(P_i)$.

2. Client→Guard Auth. Each client C_i sends $P_i, \mathcal{S}_{\hat{p}_i}(P_i), \mathcal{S}_{\hat{p}_r}(P_i)$ to all guards.

3. Shared Secrets. Each guard S_j derives n secrets $r_{ij} = \texttt{KDF}(\hat{p}_j \cdot P_i)$, one for each client with a valid signature $\mathcal{S}_{\hat{p}_r}(P_i)$ from the relay. Similarly, each client C_i derives m secrets $r_{ij} = \texttt{KDF}(p_i \cdot \hat{P}_j)$.

4. Verifiable Shuffle. Clients participate in a verifiable shuffle protocol [54] run by the guards, with the ephemeral keys $\underline{P_i}$ as input. The public output π consists of n pseudonym keys in permuted order, such that no one knows which client corresponds to which key, except the owner of the corresponding private key. More formally, we write $\pi = (\tilde{P}_{\alpha(1)}, ..., \tilde{P}_{\alpha(n)})$, where $\tilde{P}_{\alpha(i)} = c \cdot P_i$ for a permutation α and some constant c. At the end of this step, clients receive π, along with a transcript signed by all guards.

Safety Checks. In step 4, the honest guard checks that each input P_i corresponds to a client with a valid $\mathcal{S}_{\hat{p}_r}(P_i)$, or it aborts.
At the end of `Setup`, honest clients check that (1) the verifiable shuffle completed correctly, (2) π is signed by every guard in G, (3) there are at least $K = 2$ clients in T in the input, and (4) its own shuffled pseudonym is included in the permutation. If any test fails, it aborts.

Finally, the relay creates n empty dictionaries b_k, indexed by $k = \alpha(i)$, to keep track of IP sockets later used for packet forwarding.

4.2.2 Anonymize

After `Setup`, all nodes continuously run `Anonymize`. In each time slot, clients and guards participate in a DC-net protocol. All guards compute one ℓ-bit pseudo-random message from the `PRG`s seeded with the shared secrets, and send it to the relay. All clients perform likewise, except for the client owning the time slot, who additionally includes its upstream message(s) m_i in the computation. In practice, m_i is one or more IP packet(s) without source address, up to a total length ℓ. If the slot owner has nothing to transmit, it sets $m_i = 0^\ell$.

Once the relay receives the $n+m$ ciphertexts from all clients and guards, it XORs them together to obtain m_k. If the protocol is executed correctly, m_k is equal to m_i, as the values of $\texttt{PRG}(r_{ij}), i \in \{1...,n\}, j \in \{1...,m\}$ cancel out. If m_k is a full IP packet, the relay replaces the null source IP in the header by its own (just like in a NAT) and forwards it to its destination. If it is a partial packet, the relay buffers it and completes it during the next schedule.

Then, the relay broadcasts one downstream message $\underline{d}$ to all clients, each $d \in \underline{d}$ being encrypted with a public key $\tilde{P}_k \in \pi$ corresponding to an anonymous client. We emphasize that the relay does not know for which client it encrypts. Additionally, $\underline{d}$ is of arbitrary length ℓ', possibly much larger than ℓ, easily accommodating downstream-intensive scenarios. Finally, we emphasize that $\underline{d}$ can contain data for multiple users (from previous rounds). If the relay has nothing to transmit, it sends a single 0 bit to indicate the end of the round.

Protocol 2: `Anonymize`

Inputs: $\underline{r_{ij}}, \pi$, up/down-stream message sizes ℓ, ℓ'

Outputs: per round k: anonymous message m_k, downstream traffic $\underline{d}$.

For round $k \in \{1,...,n\}$:
- Each client C_i sends to the relay $c_i \leftarrow \texttt{DCNet-Gen}(\underline{r_{ij}}, x_i)$ with $x_i = \begin{cases} m_i, & \text{if } \alpha(i) = k, \quad //C_i \text{ is the sender} \\ \underline{0}, & \text{otherwise.} \end{cases}$
- Each guard S_j sends to the relay $s_j \leftarrow \texttt{DCNet-Gen}(\underline{r_{ij}}, \underline{0})$.
- The relay computes $m_k \leftarrow \texttt{DCNet-Reveal}(\underline{c_i}, \underline{s_j})$, with $m_k \in \{0,1\}^\ell$ an IP packet.
- The relay handles m_k as follows:
  - If m_k is not part of an active socket in b_k, the relay creates and stores the socket.
  - it puts its own IP address in m_k, then sends it in the appropriate socket in b_k.
- The relay computes $\underline{d} \leftarrow \texttt{Downstream}(\underline{b})$ and sends $\underline{d}$ to each client.

Function `DCNet-Gen`$(\underline{r_{ij}}, x_i)$:
  return $\bigoplus_{r \in \underline{r_{ij}}} \texttt{PRG}(r) \oplus x_i$

Function `DCNet-Reveal`$(\underline{c_i}, \underline{s_j})$:
  return $\bigoplus_i c_i \oplus \bigoplus_j s_j$

Function `Downstream`$(\underline{b})$:
  $\underline{d} \leftarrow$ array();
  for $k \in \{1,...,n\}$ **do**
    for *socket* $\in b_k$ containing downstream bytes d **do**
      add $\mathcal{E}_{\tilde{P}_k}(d)$ to $\underline{d}$
    end
  end

`Anonymize` has the following property (proved in A.2):

Property 3 [Goal G1]. After a run of `Anonymize`, let C_{i_1} and C_{i_2} be two honest clients, $k_1 = \alpha(i_1), k_2 = \alpha(i_2)$ the time slots in which they communicated, and m_{k_1}, m_{k_2} the anonymous upstream messages for those slots. Then, $\mathcal{A}$ has negligible advantage in guessing $b \in [1,2]$ such that m_{k_b} is the message sent by i_1.

4.3 Practical Considerations

End-to-End Confidentiality. A malicious relay can see the upstream message plaintexts. This is also the case for a VPN server or Tor exit relay; clients should use standard end-to-end encryption (*e.g.,* TLS) on top of PriFi.

Churn. In the case of churn, *e.g.,* if any client or guard joins or disconnects, the relay broadcasts a `Setup` request that signals the start of a new epoch. Upon reception, each node aborts and re-runs `Setup`. Churn negatively affects performance; we evaluate its effect in Section 7.5.

Bandwidth Usage. To reduce idle bandwidth usage, the relay periodically sends a "load request" in which clients can anonymously *open* or *close* their slots. The relay skips a closed slot, hence saving time and bandwidth. If all slots of a schedule are closed, the relay sleeps for a predetermined interval, further saving bandwidth at the cost of higher initial latency when resuming communications. The concrete parameters of this improvement (*e.g.,* frequency of the load requests, sleep time) are not fully explored in this work; they exhibit a typical latency-bandwidth usage trade-off.

We emphasize that load tuning does not reduce the anonymity set size; all clients still transmit ciphertexts exactly at the same time. Load tuning makes global communication volumes and packet timings more visible to an external eavesdropper, but our threat model considers a stronger, local eavesdropper (the malicious relay) who has access to this information anyway.

Finally, although this has not been investigated in this work, both up/down-stream sizes ℓ and ℓ' can be dynamically tuned without interrupting the communications. This allows the relay to better match the sending/receiving rates of the clients and further reduce idle bandwidth usage.

Synchronicity. The protocol uses message reception events, rather than clocks, to keep the participants in lock-step.

4.4 Limitations of this Protocol

Accountability. No mechanism enforces dishonest parties to correctly follow the protocol; malicious parties can anonymously disrupt the communications. This is a well-known DC-net issue [10, 14, 83], addressed in Section 5.

Downstream Anonymity. This notion refers to the clients being indistinguishable when receiving downstream messages, which is trivially the case if the relay truthfully sends the same downstream data to all clients. This property is not enforced above, but is addressed later in Section 6.

5 Disruption Protection

In the basic protocol above, a malicious active insider can modify or jam upstream communications by transmitting arbitrary incorrect bits instead of the ciphertext defined by the protocol. This is particularly problematic because the attacker is provably anonymous and untraceable.

In the related work, these attacks can be detected retroactively using "trap bit" protocols [83] that detect a disruptor with a certain probability but reduce the throughput linearly with respect to the number of trap bits. Unfortunately, the probability of detection must be high enough to detect a single bit-flip, which can effectively corrupt a message. Another technique is to rely on commitments before every DC-net message [14], which adds latency.

Some DC-nets use group arithmetic instead of binary strings, which enables proving (proactively or retroactively) that computations are correct [15, 35]. These designs do not fit low-latency requirements, unfortunately, as their computation time is significantly higher: tens of milliseconds per message for the computation alone, whereas XOR-based DC-nets take microseconds. A brief analysis of this computational cost is provided in Verdict [15] (p.12, Figure 6).

PriFi uses a retroactive, hash-based "blame" mechanism on top of a "classic" XOR-based DC-net, which (1) keeps the typical operation (in the absence of jamming) as fast as possible, and reduces the bandwidth lost due to the protection, and (2) excludes a disruptor with high probability (1/2 per flipped bit). Exploiting the LAN topology, our exclusion takes seconds, which is orders of magnitude faster than the related work [83] (see Figure B.6).

5.1 Protocol

We modify the previous `Anonymize` protocol (Protocol 2) to protect against disruption from malicious insiders. In short, we add a hash-based detection of corruption and a blame mechanism to exclude a disruptor.

Summary. The relay sends the hash of the upstream message on the downstream traffic. If the anonymous sender detects an incorrect hash, it requests a copy of its own disrupted message by setting a flag $b_{\text{echo\_last}}$ to 1.

When receiving a disrupted copy m'_k of a previously-sent message m_k, the client searches for a bit position l such that $(m_k)_l = 0$ and $(m'_k)_l = 1$. If such l exists, then the client requests to de-anonymize the l^{th} bit of his own slot k, by sending $\text{NIZKPoK}_{k,l}(\tilde{p}_k : \tilde{p}_k = \log \tilde{P}_k)$ in its next upstream message, a non-interactive proof of knowledge of the key $\tilde{p}_{(k \bmod n)}$ corresponding to slot k in π [4]. The

proof is bound to the public values l and k. For simplicity, we write $\text{PoK}_{k,l}(\tilde{p}_k : \tilde{p}_k = \log\ \tilde{P}_k)$ hereafter, thus ignoring (1) the mod n and (2) the acronym for "Non-Interactive, Zero-Knowledge".

If no such l exists, the message was disrupted but the disruptor cannot be traced without simultaneously de-anonymizing a client (see "Remarks" below for more details). In this case, nothing happens.

The `Anonymize` and `Blame` protocols are described in Protocols 3 and 4, respectively, and have the following properties (proved in Appendix A.3):

Protocol 3: `Anonymize`

(The differences with Protocol 2 are highlighted in blue.)

Inputs: $\underline{r_{ij}}, \pi, \ell, \ell'$

Outputs: per round k: m_k, $\underline{d}$.

For round $k \in \{1,...,n\}$:

- Each client C_i sends to the relay $c_i \leftarrow \texttt{DCNet-Gen}(\underline{r_{ij}}, x_i)$ with

$$x_i = \begin{cases} \underline{0}, & \text{if } \alpha(i) \neq k, \\ \text{PoK}_{k',l}(\tilde{p}_{k'} : \tilde{p}_{k'} = \log\ \tilde{P}_{k'}), & \text{if slot } k' \text{ was disrupted,} \\ m_i \| b_{\text{echo\_last}}, & \text{otherwise.} \end{cases}$$

- Each server S_j sends to the relay $s_j \leftarrow \texttt{DCNet-Gen}(\underline{r_{ij}}, \underline{0})$.
- The relay computes $m_k \leftarrow \texttt{DCNet-Reveal}(\underline{c_i}, \underline{s_j})$
- The relay handles m_k as follows:
  - if m_k is a Blame message, it starts the $\texttt{Blame}(\text{PoK}_{k',l}(\tilde{p}_{k'} : \tilde{p}_{k'} = \log\ \tilde{P}_{k'}), \underline{c_i}, \underline{s_j})$ protocol,
  - else, it sends m_k in the appropriate socket in b_k.
- The relay computes and sends to each client $\underline{d} \leftarrow \text{Downstream2}(\underline{b}, m_k, k, b_{\text{echo\_last}})$

Function $\texttt{Downstream2}(\underline{b}, m_k, k, b_{echo\_last})$**:**

$\underline{d} \leftarrow$ array();
for $k' \in \{1,...,n\}$ **do**
1. if $k' = k$: (for the anonymous sender)
   (a) add $H(m_k)$ to $\underline{d}$,
   (b) if $b_{\text{echo\_last}} = 1$: add $\mathcal{E}_{\tilde{P}_k}(m_{k-n})$ to $\underline{d}$.
2. for each socket $\in b_{k'}$ containing downstream bytes d, add $\mathcal{E}_{\tilde{P}_{k'}}(d)$ to $\underline{d}$.

end

Properties 4+5 [Goal G1]. The anonymity of any honest client is unaffected by the information made public in `Blame` (Protocol 4): $\underline{\texttt{PRG}(r_{ij})_l}$ in step 2, or r_{ij} in step 5.

Property 6 [Goal G2]. Let C_i be the owner of a slot k, and let C_d, $d \neq i$, be another client (or guard). If C_d sends an arbitrary value q instead of the value c_i (or s_j) as specified in the protocol, then C_d is identified as the disruptor and is excluded from subsequent communications.

Protocol 4: `Blame`

Inputs: $\text{PoK}_{k,l}(\tilde{p}_k : \tilde{p}_k = \log\ \tilde{P}_k), \underline{c_i}, \underline{s_j}$

1. The relay broadcasts $\text{PoK}_{k,l}(\tilde{p}_k : \tilde{p}_k = \log\ \tilde{P}_k)$ to every client/guard.
2. Each client/guard checks the PoK, and reveals the l^{th} bit of the values $\texttt{PRG}(r_{ij})$ for slot k, $\forall i \in \{1,...,n\}, j \in \{1,...,m\}$ by sending a non-anonymous, signed message $\underline{\texttt{PRG}(r_{ij})_l, \mathcal{S}_{\tilde{p}_i}(\texttt{PRG}(r_{ij})_l)}$ to the relay. A non-complying entity is identified as the disruptor.
3. For each client, the relay checks the signature, and that $\bigoplus_j \texttt{PRG}(r_{ij})_l$ indeed equals $(c_i)_l$ sent in slot k; if a mismatch is detected, this client is identified as the disruptor. The relay performs the same verification for each guard.
4. For each pair of client-guard (C_i, S_j), the relay compares $\texttt{PRG}(r_{ij})_l$ from the client and $\texttt{PRG}(r_{ij})_l$ from the guard: they should be equal. If there is a mismatch, at least one of them is lying. The relay continues by forwarding the signed message $\texttt{PRG}(r_{ij})_l$ from C_i to S_j and vice-versa.
5. C_i checks that $\underline{\texttt{PRG}(r_{ij})_l}$ is signed by S_j, and that the bit $\texttt{PRG}(r_{ij})_l$ is in contradiction with its own bit $\texttt{PRG}(r_{ij})_l$. Then, he answers with r_{ij}, the secret shared with S_j, along with a proof of correctness for computing r_{ij}. S_j proceeds similarly. A non-complying entity is identified as the disruptor.
6. The relay checks the proofs of correctness, then uses r_{ij} to recompute the correct value for $\texttt{PRG}(r_{ij})$ for slot k, identifying the disruptor.

Once the disruptor is identified, the relay excludes it from the group G and the roster T, and then broadcasts all inputs and messages exchanged in `Blame` to all clients for accountability.

Property 7. An honest entity is never identified as a disruptor.

Limitations. The detection relies on the capacity of the jammed client to transmit $b_{\text{echo\_last}}$ and the PoK; the adversary also can jam these values. In practice, l is fairly large (*e.g.,* 5 kB), and the client can use redundancy coding over his message to make the task harder for the adversary. When this probabilistic solution is insufficient, users can use a verifiable DC-net [15] to transmit without the risk of jamming; in practice, this verifiable DC-net would run in background with very low bandwidth and high latency, just enough to transmit the proof-of-knowledge.

Remarks. In step 3 of `Blame` (Protocol 4), we observe why the client starting the blame checks that $(m_k)_l = 0$: otherwise, revealing $\underline{\texttt{PRG}(r_{ij})_l}$ over a non-anonymous channel would flag this client as the sender, as $\bigoplus_j \texttt{PRG}(r_{ij})_l \neq (m_k)_l$.

In step 5 of `Blame` (Protocol 4), we remark that at least one mismatching pair exists, otherwise the slot would not have been disrupted. If multiple disruptors exist, `Blame` excludes one disruptor per disruption event.

6 Equivocation Protection

In both versions of `Anonymize` above (Protocols 2 and 3), the relay broadcasts the downstream traffic $\underline{d}$ to all clients to ensure receiver anonymity. However, no mechanism enforces truthful broadcast, so a malicious relay can perform *equivocation attacks*: *i.e.,* send different messages to each client, hoping that their subsequent behaviors will reveal which client actually decrypted the message. This attack can be seen as a "poisoning" of downstream traffic.

Equivocation Example. Clients C_1 and C_2 are both honest. On the first round, the relay decodes an anonymous DNS request. Instead of broadcasting the same DNS answer to C_1 and C_2, the relay sends two different answers containing IP_1 and IP_2, respectively. Later, the relay decodes an anonymous IP packet with IP_2 as destination. It can guess that C_2 made the request, as C_1 has never received IP_2.

In practice, a credible scenario is a router infected with malware or compromised by the adversary and spying on honest users of a corporate network, possibly colluding with the endpoints contacted by the clients.

We note that previous DC-net systems do not mention this issue [14, 15, 83]. The attack is possible because a malicious party relays the information, which can happen in both Dissent [83] and Verdict [15]. If the traffic is unencrypted, the attack is trivial. The use of higher-level encryption (*e.g.,* TLS) can offer a mitigation, *if* we further assume that the remote endpoint does not collude with the malicious relay. In practice, having two particular entities under the control of the adversary (the relay and some interesting external service, *e.g.,* WikiLeaks) does not seem impossible, however.

On the contrary, we note that due to their different design, mix-nets and onion-router networks are typically not affected by this attack.

PriFi Solution. Intuitively, to thwart an equivocation attack, clients need to agree on what they have received before transmitting their next message. In PriFi, this is achieved without adding extra latency and without synchronization between clients. Clients encrypt their messages before anonymizing them; the encryption key depends on the history of downstream messages and also on the shared secrets with the guards. The relay is thus unable to recover a plaintext if not all clients share the same history.

6.1 Protocol

We modify the previous `Anonymize` and `Blame` protocols (Protocol 5 and 6). These are the final variants used in our implementation (Section 7).

`Anonymize`. Each client C_i keeps a personal history h_i of downstream communications. Upon receiving a downstream message $\underline{d}$, each client updates its history.

Each upstream message is then symmetrically-encrypted with a fresh key γ. This value γ is sent to the relay, blinded by a function of the downstream history h_i. Only if all honest clients agree on the value h_i, the relay can unblind γ and decrypt the message.

`Blame`. The previous `Blame` protocol finds a disruptor when values c_i or s_j are not computed correctly; we add a way to validate the new values κ_i and σ_j. As they are elements of $\mathbb{G}$, we apply a standard zero-knowledge proof [4]. More precisely, we use an Or/And type of NIZKPoK which allows the clients to prove either (1) their ownership of the slot or (2) that κ_i is computed correctly.

The `Anonymize` and `Blame` protocols have the following properties (proved in Appendix A.4):

Properties 8+9+10+11 [Goal G1]. The anonymity of any honest client is unaffected by the extra information revealed: κ_i in step 3 of `DCNet-Gen-Client`, σ_j in step 2 of `DCNet-Gen-Guard`, $\underline{Q_i}$,PoK in step 7 of `Blame`, or r_{ij} in step 9 of `Blame`.

Property 12 [Goal G1]. If $\exists i, j$ two honest clients who received $\underline{d_i} \neq \underline{d_j}$ on round k, then neither the relay nor $\mathcal{A}$ can decrypt m_k for any subsequent round $k' > k$.

Property 13 [Goal G2]. If a client C_i sends an arbitrary value κ_i' instead of the value κ_i as specified in the protocol, then C_i is identified as the disruptor and is excluded from subsequent communications.

Properties 14+15. Honest parties are not blamed for equivocation attacks or for disruption attacks.

6.2 Practical Considerations

Packet Losses. We note that this protection is decoupled from link-layer retransmissions; if one client fails to receive a packet, it will ask the relay again after a timeout, delaying all clients for this specific round (which is unavoidable for traffic-analysis resistance) but not invalidating the whole epoch with a wrong h_i value.

Protocol 5: `Anonymize` (final version)

(The differences with Protocol 3 are highlighted in blue.)

Inputs: $\underline{r_{ij}}, \pi, \ell, \ell'$

Outputs: per round k: m_k, $\underline{d}$.

For round $k \in \{1,\ldots,n\}$:

- Each client C_i sends to the relay $c_i, \kappa_i \leftarrow$ `DCNet-Gen-Client`$(\underline{r_{ij}}, m_i, h_i)$
- Each guard S_j sends to the relay $s_j, \sigma_j \leftarrow$ `DCNet-Gen-Guard`$(\underline{r_{ij}})$
- The relay computes $m_k \leftarrow$ `DCNet-Reveal2`$(\underline{c_i}, \underline{s_j}, \underline{\kappa_i}, \underline{\sigma_j})$ with $m_k \in \{0,1\}^l$ or $\perp$.
- The relay handles m_k as follows:
  - if m_k is a Blame message, it starts the `Blame` protocol,
  - else, it sends m_k in the appropriate socket in b_k.
- The relay outputs $\underline{d} \leftarrow$ `Downstream2`$(\underline{b}, H(m_k), k, b_{\text{echo\_last}})$
- The relay updates $h_r \leftarrow H(h_r || \underline{d})$, and sends $\underline{d}, \mathcal{S}_{\tilde{p}_r}(h_r)$ to each client.
- When receiving $\underline{d}$, each client checks the signature $\mathcal{S}_{\tilde{p}_r}(h_r)$ or aborts. Then, each client C_i updates $h_i \leftarrow H(h_i || \underline{d})$.

Function `DCNet-Gen-Client`$(\underline{r_{ij}}, m_i, h_i)$**:**

1. compute x_i as follows:
   - if $\alpha(i) = k$:
     - if slot k' was disrupted, $x_i \leftarrow \text{PoK}_{k,l}(\tilde{p}_k : \tilde{p}_k = \log \tilde{P}_k)$
     - else, pick a random symmetric key $\gamma \xleftarrow{\$} \{0,1\}^\lambda$, compute $m'_i = \mathcal{E}_\gamma(m_i)$, and set $x_i \leftarrow m'_i || b_{\text{echo\_last}}$
   - else, $x_i \leftarrow \underline{0}$
2. compute $c_i \leftarrow \bigoplus_i \texttt{PRG}(r_{ij}) \oplus x_i$
3. compute κ_i as follows:
   - if $\alpha(i) = k$: $\kappa_i \leftarrow F_1(\gamma) + F_2(h_i) \cdot \sum_{j=1}^{m} F_3(H(\texttt{PRG}(r_{ij})))$
   - else, $\kappa_i \leftarrow F_2(h_i) \cdot \sum_{j=1}^{m} F_3(H(\texttt{PRG}(r_{ij})))$

return c_i, κ_i

Function `DCNet-Gen-Guard` $(\underline{r_{ij}}, x_i)$**:**

$s_j \leftarrow \bigoplus_i \texttt{PRG}(r_{ij})$

$\sigma_j \leftarrow -\sum_{i=1}^{n} F_3(H(\texttt{PRG}(r_{ij})))$

return s_j, σ_j

Function `DCNet-Reveal2`$(\underline{c_i}, \underline{s_j})$**:**

$m'_k \leftarrow \bigoplus_i c_i \oplus \bigoplus_j s_j$

$F_1(\gamma) \leftarrow F_2(h_r) \cdot \sum_{j=1}^{m} \sigma_j + \sum_{i=1}^{n} \kappa_i$

$m_k \leftarrow \mathcal{D}_\gamma(m'_k)$

return m_k

Protocol 6: `Blame` (final version)

(The differences with Protocol 4 are highlighted in blue.)

Inputs: epoch ID e, $\text{PoK}_{k,l}(\tilde{p}_k : \tilde{p}_k = \log \tilde{P}_k), \underline{c_i}, \underline{s_j}$

1. The relay broadcasts $\text{PoK}_{k,l}(\tilde{p}_k : \tilde{p}_k = \log \tilde{P}_k)$ to every client/guard,
2. Each client/guard checks the PoK and sends to the relay $\underline{\texttt{PRG}(r_{ij})_l}, \mathcal{S}_{\tilde{p}_i}(e, \underline{\texttt{PRG}(r_{ij})_l})$ for slot k, $\forall i \in \{1,\ldots,n\}, j \in \{1,\ldots,m\}$.
3. For each client/guard, the relay checks the signature, and that $\bigoplus_j \texttt{PRG}(r_{ij})_l$ indeed equals $(c_i)_l$ sent in slot k.
4. For each pair of client-guard (C_i, S_j), the relay compares $\texttt{PRG}(r_{ij})_l$ from the client and $\texttt{PRG}(r_{ij})_l$ from the guard. If there is a mismatch, the relay forwards the signed message $\underline{\texttt{PRG}(r_{ij})_l}$ from C_i to S_j and vice-versa.
5. C_i checks that the signature and that the bit $\texttt{PRG}(r_{ij})_l$ mismatches with its own bit $\texttt{PRG}(r_{ij})_l$. Then, it answers with r_{ij} along with a proof of correctness. S_j proceeds similarly.
6. The relay checks the proofs, then uses r_{ij} to recompute the correct value for $\texttt{PRG}(r_{ij})$ for slot k. If no disruptor is identified, the relay continues.
7. Each client C_i computes m values $Q_i = P \cdot F_3(H(\texttt{PRG}(r_{ij}))), \forall j \in \{1,\ldots,m\}$, where P is the base point of $\mathbb{G}$. Then, each client computes a PoK as follows:

$$\text{PoK}\Big\{\tilde{p}_k, q, q_1 \ldots, q_m : \tilde{p}_k = \log \tilde{P}_k \vee \{ q = \log \kappa_i \wedge q_1 := \log Q_1 \wedge \vdots \; q_m := \log Q_m \wedge q := q_1 + \cdots + q_m \}\Big\}$$

   Each client sends a message $\underline{Q_i}, \text{PoK}, \mathcal{S}_{\tilde{p}_i}(e, \underline{Q_i}, \text{PoK})$ to R. A non-complying entity is identified as the disruptor. Each server performs similarly, minus the first clause of the PoK which is never true.
8. The relay checks all signatures and PoKs, and potentially identifies a disruptor. If not, for each pair of client-guard (C_i, S_j), the relay compares the two values Q_i, Q_j — they should be equal. If there is a mismatch, at least one of them is lying. The relay continues by sending the signed message $\underline{Q_i}, \text{PoK}, \mathcal{S}_{\tilde{p}_i}(e, \underline{Q_i}, \text{PoK})$ from C_i to S_j and vice-versa.
9. C_i checks the signature and the PoK, and that a value Q_i is in contradiction with some value of its own. Then, it answers with r_{ij}, the secret shared with S_j, along with a proof of correctness for computing r_{ij}. S_j proceeds similarly. A non-complying entity is identified as the disruptor.
10. The relay checks the proofs of correctness, then uses r_{ij} to recompute the correct values for $\texttt{PRG}(r_{ij})$ and $H(\texttt{PRG}(r_{ij}))$ for slot k, identifying the disruptor.

Once the disruptor is identified, the relay excludes it from the group G and the roster T, and then broadcasts all inputs and messages exchanged in `Blame` to all clients for accountability.

7 Evaluation

This evaluation is abridged for space; for the full evaluation see the extended version [3].

We implemented PriFi in Go [62]. We evaluate it on a LAN topology typical of a small organizational network.

Methodology. Our evaluation is five-fold. First, we measure the end-to-end latency via a SOCKS tunnel without

data, by having a client randomly ping the relay. Second, we compare PriFi with prior DC-nets. Third, we replay network traces representing realistic workloads on PriFi, and measure the added latency and bandwidth usage. Fourth, we explore limits of the system by evaluating two alternative deployment scenarios: having a local trusted guard, which will be relevant in the case of the ICRC, and having clients outside of the LAN. Finally, we explore the effect of churn and user mobility on PriFi.

Experimental Setup. We use Deterlab [25] as a testbed. The experimental topology consists of a 100Mbps LAN with 10ms latency between the relay and the clients. We run three guards, each having a 10Mbps link with 100ms latency to the relay. We use nine machines, one dedicated to the relay and one per guard. The clients are simulated on the remaining five machines, distributed equally. All machines are 3GHz Xeon Dual Core with 2GB of RAM. We focus our evaluation between 2 and 100 users, which is inspired by the ICRC's operational sites (Figure B.4).

Security Parameters. We rely on the Kyber cryptographic library [22]. We use $\lambda = 256$ bits, Curve25519 for $\mathbb{G}$, SHA-256 as a hash function, and Schnorr signatures. The DC-net PRG uses BLAKE2 as an extensible output function.

Reproducibility. All experiments presented in this paper are reproducible with a few simple commands after cloning the repository [62]. All raw logs and scripts to recreate the plots are available in a separate repository [63], with the exception of the private ICRC dataset.

7.1 End-to-End Latency without Data

Figure 3(a) shows the latency of the PriFi system, *i.e.,* the time needed for an anonymized packet to be sent by the client, decoded by the relay, and sent back to this same client. In this experiment, one random user is responsible for measuring these "pings", whereas others only participate in the protocol without sending data (*i.e.,* the number of active users is 1, anonymous among all users). We observe that the latency increases linearly with the number of clients, from 58ms for 20 users (*e.g.,* a small company) to 86ms for 100 users, and it scales reasonably well with the number of clients. We also observe that a major component of the latency is the buffering of messages by the clients; with only one slot per schedule, clients must wait for this slot before transmitting data.

Pipelining. To reduce further latency, we *pipeline* rounds: we run multiple DC-net rounds in parallel instead of the "ping-pong" presented in `Anonymize`. This enables us to better utilize the available bandwidth and reduce latency, until the capacity of the links is reached. In this experiment, this further divides the latency by 2.25 (Figure B.5).

Pipelining and Equivocation Protection. These two components have a subtle interaction: at any point in time, the equivocation protection is computed with all the *received* data, naturally ignoring "in-flight" data from the relay to the clients. Pipelining increases the amount of in-flight data. Importantly, this is done without packet reordering. Each message sent by the clients depends on the same received rounds, benefiting correctly from equivocation-protection.

CPU. Finally, during the same experiment, we briefly evaluate the CPU and memory cost on the relay (Figure B.7).

7.2 Comparison with Prior DC-Net Designs

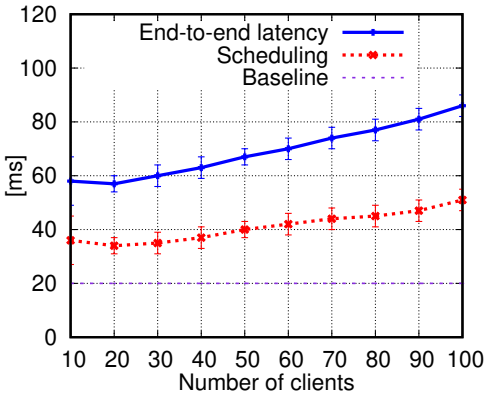
Benchmarking. We select two related works for comparison. The closest is Dissent in Numbers (abbreviated D#) [83]. Like PriFi, D# provides provable traffic-analysis by using binary-string-based DC-nets, has similar assumptions (M anytrust servers) but with no particular emphasis on being low-latency. We then compare with a more recent ACN, Riffle [41]: it has a similar topology but emphasizes on minimizing the download bandwidth for clients. We do not compare with mix-nets and onion-routing protocols, whose architecture is significantly different, in that users' messages are routed *sequentially* through multiple hops over the Internet.

The first major difference between PriFi and both Riffle and D# is in the functionality of the guards: Both protocols require several server-to-server communications per round before outputting any anonymized data.

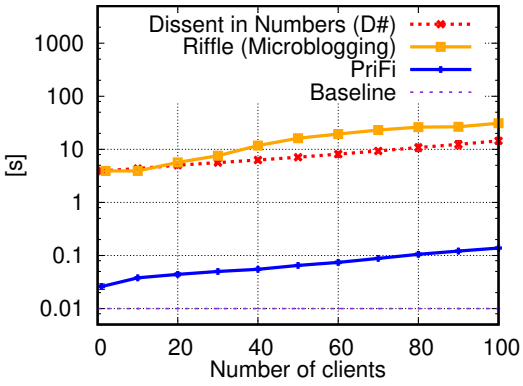
We deploy Riffle and D# on our setup and compare their latency against PriFi in Figure 3(b). For 100 users, a round-trip message takes $\approx$14.5 s in D# (excluding setup), $\approx$31 seconds in Riffle (which includes a one-time setup cost of $\approx$7.3 seconds), and 137ms in PriFi (excluding setup).

Higher-Level Comparison. We also numerically compare PriFi against Riposte [16], a PIR-based protocol, and DiceMix [66], a group-arithmetic-based DC-net.

Riposte uses expensive cryptography to save bandwidth. As a result, Riposte can process up to 2 messages/sec with 2 servers (Fig. 7, p.16 of [16]), whereas our relay outputs hundreds of messages/sec (Figure 3(a)). The topology of Riffle is precisely what we avoid in PriFi. In Riffle, the anytrust servers sequentially decrypt the client ciphertexts, then the last server broadcasts the results to all



(a) End-to-end latency experienced by any client. The baseline is twice the latency of the LAN (20 ms).



(b) Latency comparison between PriFi, D# and Riffle, computed as the time to send and decode an anonymous message. The baseline is the latency of the LAN (10 ms).

Fig. 3. End-to-end latency and comparison with the related work.

servers before any client can download it, thus increasing latency on the critical transmission path (Figure 1).

To keep a low-latency, PriFi does not use group-arithmetic-based DC-nets like DiceMix [66]. DiceMix's latency is in the order of seconds (≈20s for 100 participants; Fig. 4 p.10 of [66]). Although these DC-nets allow for better collision resistance and proofs of correct computation, when we evaluated them in PriFi, we found out that the generation of pseudo-randomness alone was too slow for low-latency communications. This problem was also highlighted in Fig. 6, p.12 of Verdict [15].

7.3 Latency with Recorded Traffic Datasets

CRAWDAD Traces. We evaluate the performance of PriFi when replaying the dataset 'apptrafictraces' [69] from CRAWDAD [17]. We selected two sub-traces: a 'Skype' trace where one client performs a VoIP (non-video) call for 281 seconds, and a 'Hangouts' trace where one client performs a video call for 720 seconds.

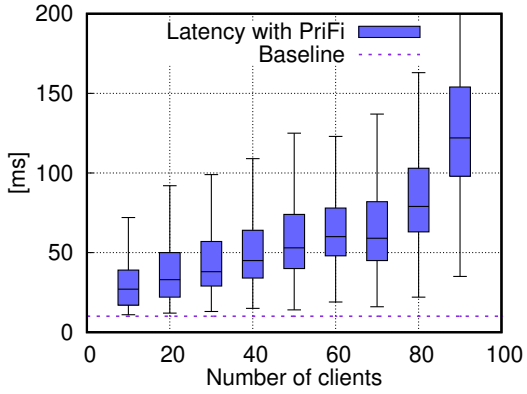
Using the same setup as before, 5% of the clients are randomly assigned packet traces from a pool and, after a random delay $r \in [0,30]$ seconds, they replay the packets through PriFi. The relay decodes the packets and records the time difference between the decoded packet and the original trace. Because most endpoints in the traces were not reachable anymore on today's Internet, the recorded latency does not include the communication to the Internet endpoints, but only the latency added by PriFi.

ICRC Traces. We also replay a dataset recorded at a ICRC delegation from June to July 2018. The capture contains network-level packet headers only, corresponding to all network traffic for 30 days of capture. During active periods, corresponding to work days, the mean number of users is 60.9, with a standard deviation of 5.8. Also during active periods, the mean bitrate of the network is of 3.1 Mbps, with a standard deviation of 4.7 and a (single) peak at 25 Mbps corresponding to a bulk file transfer. To evaluate PriFi with this dataset, we first randomly select 10 1-hour periods from the active periods (*i.e.,* we exclude weekends and nights); we replay those 10 sub-traces and measure the latency and bandwidth overhead. During this hour, we simulate a varying number of clients: First, we identify (and only simulate) local clients, identified by an IP address of the form 10.128.10.x; these clients replay their own packets. Second, when needed, we add additional clients who represent extra idle users. These clients send no payload data but increase the anonymity set size. We average the results over the 10 1-hour periods and over all clients.

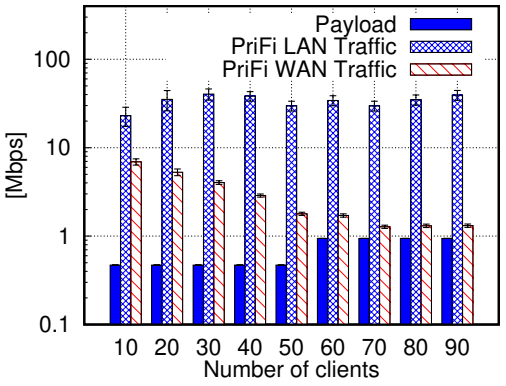
Analysis. Figures 4(a) and 5(a) show the added latency on the 'Skype' and 'Hangouts' datasets. Figure 4(b) shows the bandwidth used; PriFi has similar bandwidth usage for all three datasets. The latency increases with the number of clients, which is due to (1) the increasing traffic load going through PriFi, as more users send data, and (2) to the increasing time needed to collect all clients' ciphertexts.

In Figure 4(b), we show the bandwidth used by the system, split into two components: the bandwidth used in the LAN, and in the WAN. We also show the bitrate of the payload, as an indication of the useful throughput (goodput) of the system. We see that the LAN bandwidth usage is typically around 40 Mbps, whereas the WAN usage varies from 6.9 to 1.3 Mbps. We recall that, in an organizational setting, the bandwidth of the LAN is typically 100 Mbps or 1 Gbps, and that the bottleneck typically lies on the link towards the Internet.

In Figure 4(b), we see that the WAN bandwidth usage decreases with the number of clients. This is a shortcoming that indicates that PriFi spends more time waiting and

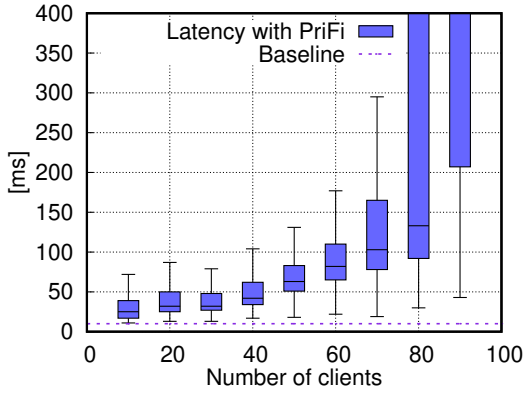


(a) Latency increase when using PriFi. The baseline is the latency of the LAN (10 ms).

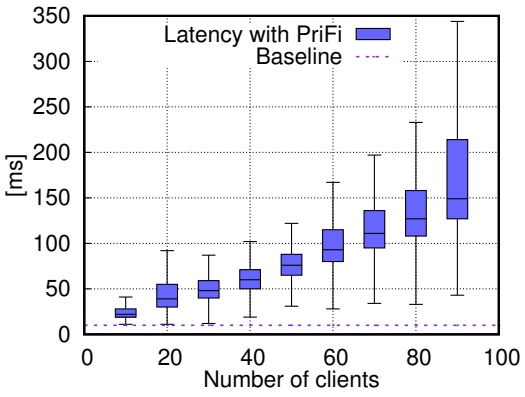


(b) PriFi bandwidth usage. The total bandwidth usage is the sum of last two 2 bars (the payload is included in the LAN traffic). The available bandwidth in the LAN is 100 Mbps.

Fig. 4. PriFi performance when 5% of the users perform a Skype call. The remaining 95% of the users are idle.



(a) 5% of users performing a Google Hangout video call. The latency drastically increases at 80 clients, indicating the limits of the current implementation.



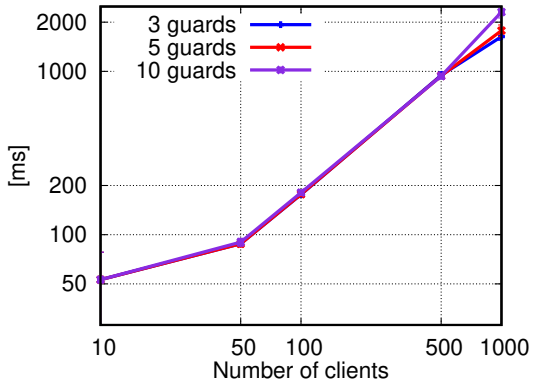
(b) Latency overhead when replaying the 'ICRC' dataset, with 100% of users having realistic activity.

Fig. 5. PriFi latency with the dataset 'Hangouts' and 'ICRC'. In both cases, the bandwidth usage (not shown here) is similar to the one observed in Figure 4(b) (except for the payload). In both cases, the baseline is the latency of the LAN (10 ms).

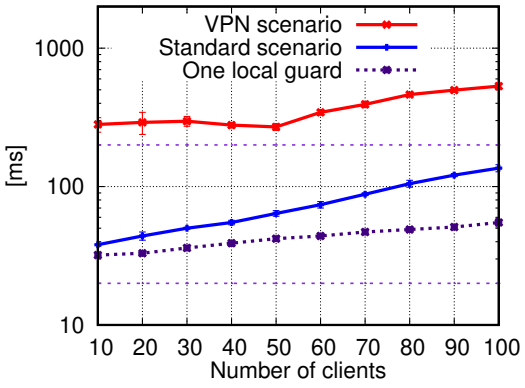
less time transmitting, due to the increased time needed to collect ciphers from more clients. This means that PriFi cannot fully utilize the available bandwidth to offer minimal latency. This could be mitigated by increasing the pipelining of rounds for slow clients so that all clients answer in a timely fashion.

We learn the following: First, the mean added latency in the case of a Skype call (with 5% active users) is below 100ms for up to 80 clients, and below 150ms for 100 clients. The International Telecommunication Union estimates that the call quality starts degrading after a 150ms one-way latency increase [76], and users start noticing a degraded quality after a 250ms one-way latency increase [78]. Hence, the current implementation supports VoIP calls for 0–80 users and reaches its limits at around 100 clients. Second, the replay of the 'Hangouts' data exhibits similar behavior as the 'Skype' dataset; we see that the latency increases reasonably until 70 users, but then drastically increases: the current implementation cannot transmit the data fast enough and buffering occurs at the clients.

When replaying the ICRC traces, shown in Figure 5(b), we observe that the added latency varies between 15 and 147 ms. This experiment was conducted with clients having network traffic corresponding to the real workload of the ICRC network. In addition, extra idle clients were simulated to achieve a constant anonymity set size of 100. This result confirms that PriFi can handle a realistic workload in the case of an ICRC delegation. We emphasize that all traffic has been anonymized through the same PriFi network, regardless of QoS. In practice, large file transfer (*e.g.,* backups) would probably either be excluded from a low-latency network, or anonymized through other means (*e.g.,* PriFi configured to provide higher throughput at higher latency).



(a) Larger-scale scalability. This experiment's purpose is to understand the limits of the system.



(b) Latencies in the three scenarios. The baseline for the VPN scenario is 200 ms, and 20 ms in the two other scenarios.

Fig. 6. Scalability and performance in various scenarios.

7.4 Scalability & Different Scenarios

Scalability. Figure 6(a) shows the performance for larger anonymity sets and with more guards. While the number of guards has almost no impact on performance due to the buffering at the relay, our implementation would not be low-latency for more than a few hundred clients.

Local Trusted Guard. The ICRC benefits from the particular situation of *Privileges and Immunities* (P&I) [21], legal agreements with governments that provide a layer of defense in order to operate in environments of armed conflicts and other situations of violence. In practice, P&I notably grants the delegations with inviolability of premises and assets. Together with the strong physical security deployed at their server rooms, each delegation essentially has a local trusted server. Aside from the ICRC, P&I can apply to embassies and diplomatic missions [23].

We simulate this new deployment with one guard in the LAN instead of three remote guards. The latency between the relay and the unique guard is 10 ms. In this case, we observe that the latency experienced by clients is roughly cut in half, shown in Figure 6(b), purple dotted curve versus blue solid curve. An additional benefit is that the only WAN bandwidth usage is the anonymized goodput.

VPN.@ When a member of an organization is accessing the network remotely (*e.g.,* when traveling), it can benefit from PriFi's protection from outside the organizational LAN. The cost is in performance, as the relay waits upon the slowest client to decode an anonymous message.

We simulate this alternative deployment scenario by having *all* clients outside the LAN. This is modeled by setting the latency between clients and relay to 100 ms instead of 10 ms. In this case, the baseline for latency is 200 ms. We observe that in this scenario, end-to-end latency varies from 280 ms to 498 ms as shown in Figure 6(b), red solid curve versus blue solid curve. While this latency is too high to support VoIP and videoconferencing, it might be acceptable for web browsing. We note that all users are slowed down. Although not explored in this work, this slowdown could be mitigated by having two PriFi networks, one reserved for local users, the other accepting remote users. Local users would participate in both networks, ensuring a sufficiently large anonymity set, and would communicate only using the fastest PriFi network.

Table 1. Effect of different churn handling strategies on PriFi.

Strategy	# Interrupt.	Availability [%]	Max Downtime [s]
Naïve	254	98.73	1.55
Abrupt	32	99.82	0.82

Loss Rates. We briefly explore the impact of various loss rates in Figure B.1. While an imperfect representation, this experiment could sketch the performance in a real WLAN. The results show that the current implementation requires "good" link quality (loss rates $\leq 10\%$, see [71] Figure 3a) to maintain low latency, that then degrades rather quickly with increasing loss rates. We note that current WLANs typically have good resilience to message drops; noise and collisions result in increased jitter rather than losses [39]. Implementing PriFi directly on Network Interface Cards (NIC) could give better control and performance. Finally, we note that WLANs have a less expensive broadcast than LANs, a factor not reflected in this experiment.

7.5 Client Churn

This is a shortened version; the details are available in the extended version of the paper [3].

In DC-nets, churn invalidates the current communications and leads to data re-transmissions and global downtime where no one can communicate. Although

re-transmissions are acceptable with PriFi's small and frequent rounds (*e.g.,* a few 100KB of payload each 10ms), frequent churn could prevent delay-sensitive applications from running on top of PriFi. To our knowledge, our contribution here is the first analysis of the impact of churn on DC-nets in a realistic scenario where nodes are mobile (*e.g.,* wireless devices).

Dataset. To characterize node mobility, we use a standard dataset [59] from CRAWDAD. The dataset contains 4 hours of traffic recorded in a cafeteria, and includes the devices' association and disassociation requests. It has 254 occurrences of churn consisting of 222 associations (33 unique devices) and 32 disconnections (12 unique devices).

Dataset Analysis. Each device (dis)connection induces a re-synchronization time of D milliseconds (for `Setup`), where D is dominated by the number of guards m and clients n and the latency between them. A typical value for D would be a few seconds (Figure B.8). We analyze two strategies to handle churn:

- The *naïve* approach stops communication for D seconds unconditionally at every churn event.
- An *abrupt* disconnection stops communication for D seconds at each disconnection only. Devices connect using a graceful approach where `Setup` is done in the background, keeping the previous `Anonymize` protocol running until the new set of clients is ready. This strategy can be enforced by the relay.

We display the results in Table 1. Notably, using the best strategy, the longest disconnection period is 0.82s; this might be noticeable as a slight lag by users of time-sensitive applications (*e.g.,* VoIP).

Anonymity Metrics. We now analyze the size of the anonymity set with respect to time, *i.e.,* among how many participants a PriFi user is anonymous at any point in time (Figure B.2). In particular, we are interested in the variations, which are due to user mobility in this case.

We display the difference, in percentage, between the actual anonymity set size and the baseline tendency. We see that size of the anonymity set does not vary more than ±8%, and the mean number of users is 50. Hence, our estimation for the worst-case of "anonymity loss" in this scenario is of 4 users, which seems acceptable in an anonymity set of 50 users.

8 Discussion on Intersection Attacks

Goal G1 ensures that a client's message is anonymous among other honest participating members. In the case where a client performs the same recognizable action (e.g., contacting a particular website) in two different epochs, the adversary can guess that this anonymous client is the same in the two epochs, and the anonymity set of this client is reduced. This problem can be exacerbated if these recognizable actions are performed over a long period of time. Also, to facilitate intersection attacks, the malicious relay could actively kick out clients. PriFi does not have a perfect solution to the problem, but we suggest the following approaches.

First, in the context of an organization, desktop computers should be connected to PriFi most of the time, providing a baseline anonymity set.

Second, as is the case with Tor, users should ideally be cautious of "blending in" by having traffic patterns similar to other users. This could be partially enforced by the organization by having a standard set of applications. If the destinations are sensitive (*e.g.,* a specific website regularly visited by one client), all clients could periodically make requests to random recently-contacted websites, for which a public, signed list would be broadcast by the relay. Incidentally, this would improve deniability *and* accountability: knowing that the list of destinations is public, an employee is less inclined to misbehave.

More importantly, clients should communicate only when a sufficiently large number of users are online, for example by using a system such as Buddies [85] and communicating only when their trusted "friends" are online. Considering that in an organization's building, most people roughly share the same working hours, we postulate that this requirement is not too constraining.

We note that traditional approaches such as anonymous authentication protocols (*e.g.,* DAGA [73]) seem fundamentally insufficient, as the clients are potentially directly connected to the malicious relay, which could actively fingerprint them using a plethora of techniques [24, 30, 37]. In this case, a segregated network topology could make the task harder for the adversary, but could increase the cost of broadcasting downstream messages.

9 Related Work

One straightforward way to protect against local eavesdroppers is by tunneling the traffic through a VPN that is outside of the adversary's control. However, this provides no

guarantee when the VPN provider is malicious. Moreover, VPNs protect neither communication patterns nor, in the case of a local eavesdropper, the communication source.

Anonymous Communication Networks (ACN) are the closest related work, but existing solutions translate poorly to the LAN setting:

Onion Routing. Tor [26] does not provide traffic analysis resistance against a global passive adversary [55, 56, 58, 79, 80]. Some prior work focuses on low latency and efficiency, at the cost of traffic-analysis resistance (LAP [38], Dovetail [67], Hornet [11]). Taranet provides traffic-analysis resistance through traffic shaping [12]. Ricochet [7] and Pond [45] are messaging systems that build on Tor.

Mix Networks. Older solutions such as Crowds [64], Mixminion [19], and Tarzan [31] are not traffic-analysis resistant. Although more recent work addresses this issue, due in part to the Anonymity Trilemma [20], this incurs high costs in either latency or bandwidth. A common way to control these costs is by having application-*specific* ACNs: Vuvuzela [77], Atom [42], Karaoke [47], Loopix [60], Stadium [75], XRD [43] for messaging/microblogging/tweets; Herd [50], Yodel [48] for VoIP; Herbivore [34], Aqua [49] for file sharing.

Both onion-routing and mix-networks are sub-optimal in providing anonymity in an organizational network. Servers should typically be non-colluding, and having them all in the same LAN reduces collective trustworthiness. When the servers are outside the LAN, both designs route users' traffic over the Internet, adding latency.

Differential Privacy. Differential privacy [28] allows leaking a small amount of statistical information about the user's communication. These systems are typically built on mixnets [46, 47, 75, 77].

PIR. A category of ACN relies on PIR [13] and ORAM [6] to implement efficient messaging systems (Riposte [16], Riffle [41], Pung [1]). These are typically not made for low latency, as the anonymity set is built over a time period.

SDN. Some solutions use Software Defined Networks to anonymize packet headers, but they are typically not traffic-analysis resistant against active attacks [53, 72, 88].

DC-nets. DC-nets [10, 14, 83] have provable traffic-analysis resistance and typically have high bandwidth and latency costs. PriFi fits this category and focuses on low latency in the context of organizational networks. Unlike the binary-string-based DC-nets used in PriFi, some related works rely on group arithmetic (Verdict [15], [35], DiceMix [66]), which enables computations to be proven correct. Unfortunately, their high computational cost makes them unsuited for low-latency applications.

DC-nets have been further studied with an emphasis on collision resolutions [32, 33] and user scheduling [40].

PriFi16. PriFi builds upon PriFi16 [2], that sketched the idea of DC-nets for LANs. Despite considering malicious insiders, PriFi16 does not provide a solution to insider jamming. PriFi16 acknowledges the problem of the equivocation without providing a concrete solution.

Summary. PriFi is a low-latency, traffic-agnostic solution working like a VPN, conceptually close to Tor [26], Hornet [11] and Taranet [12], but tailored for LANs/WLANs.

10 Conclusion

We have presented PriFi, an anonymous communication network that protects organizational networks from eavesdropping and tracking attacks. PriFi exploits the characteristics of (W)LANs to provide low-latency, traffic-agnostic communication.

PriFi reduces the high communication latency of prior work via a new client/relay/server architecture. This new architecture removes costly server-to-server communications, and allows client's traffic to be decrypted locally, remaining on its usual network path. This avoids the latency bottleneck typically seen in other systems.

PriFi also addresses two shortcomings of the related work: First, users are protected against equivocation attacks without added latency or costly gossiping; second, leveraging the LAN topology, disruption attacks are detected retroactively and orders of magnitude faster.

We have implemented PriFi and evaluated its performance on a realistic setup, mimicking the targeted ICRC deployment. Our findings show that various workloads can be handled by PriFi, including VoIP and videoconferencing, and that restrictions usually imposed by DC-nets in case of churn when users are mobile are not problematic in PriFi.

Acknowledgments. We are thankful to Vincent Graf Narbel, Alejandro Cuevas, Caleb Malchik, Jun Chen, Lucas Gauchoux, Matthieu Girod, Pierre Sarton, Yannick Schaeffer and Julien Weber for their valuable help and feedback on this project.

This research was supported in part by U.S. National Science Foundation grants CNS-1407454 and CNS-1409599, U.S. Department of Homeland Security grant FA8750-16-2-0034, U.S. Office of Naval Research grants N00014-18-1-2743 and N00014-19-1-2361, and by the AXA Research Fund.

References

[1] S. Angel and S. Setty. Unobservable communication over fully untrusted infrastructure. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 551–569, 2016.

[2] L. Barman, M. Zamani, I. Dacosta, J. Feigenbaum, B. Ford, J.-P. Hubaux, and D. Wolinsky. PriFi: A low-latency and tracking-resistant protocol for local-area anonymous communication. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*, pages 181–184. ACM, 2016.

[3] L. Barman, I. Dacosta, M. Zamani, E. Zhai, B. Ford, J.-P. Hubaux, and J. Feigenbaum. PriFi: Low-latency metadata protection for organizational network (extended version). https://arxiv.org/abs/1710.10237, 2020.

[4] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 103–112. ACM, 1988.

[5] D. Boneh. The decision diffie-hellman problem. In *International Algorithmic Number Theory Symposium*, pages 48–63. Springer, 1998.

[6] D. Boneh, D. Mazieres, and R. A. Popa. Remote oblivious storage: Making oblivious RAM practical, 2011. *Technical Report*, 2011.

[7] J. Brooks et al. Ricochet: Anonymous instant messaging for real privacy, 2016. https://ricochet.im.

[8] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 605–616. ACM, 2012.

[9] Y.-C. Chang, K.-T. Chen, C.-C. Wu, and C.-L. Lei. Inferring speech activity from encrypted skype traffic. In *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pages 1–5. IEEE, 2008.

[10] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology*, 1(1): 65–75, 1988.

[11] C. Chen, D. E. Asoni, D. Barrera, G. Danezis, and A. Perrig. Hornet: high-speed onion routing at the network layer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1441–1454. ACM, 2015.

[12] C. Chen, D. E. Asoni, A. Perrig, D. Barrera, G. Danezis, and C. Troncoso. Taranet: Traffic-analysis resistant anonymity at the network layer. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 137–152. IEEE, 2018.

[13] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 41–50. IEEE, 1995.

[14] H. Corrigan-Gibbs and B. Ford. Dissent: accountable anonymous group messaging. In *CCS*, pages 340–350, 2010.

[15] H. Corrigan-Gibbs, D. I. Wolinsky, and B. Ford. Proactively accountable anonymous messaging in Verdict. In *USENIX Security*, 2013.

[16] H. Corrigan-Gibbs, D. Boneh, and D. Mazieres. Riposte: An anonymous messaging system handling millions of users. In *IEEE Security and Privacy*, 2015.

[17] CRAWDAD. A community resource for archiving wireless data at dartmouth. http://crawdad.org/, 2016.

[18] G. Danezis and A. Serjantov. Statistical disclosure or intersection attacks on anonymity systems. In *International Workshop on Information Hiding*, pages 293–308. Springer, 2004.

[19] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a type iii anonymous remailer protocol. In *2003 Symposium on Security and Privacy, 2003.*, pages 2–15. IEEE, 2003.

[20] D. Das, S. Meiser, E. Mohammadi, and A. Kate. Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency-choose two. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 108–126. IEEE, 2018.

[21] E. Debuf. Tools to do the job: The ICRC's legal status, privileges and immunities. https://www.icrc.org/en/international-review/article/tools-do-job-icrcs-legal-status-privileges-and-immunities, 2016.

[22] DEDIS. Kyber. https://github.com/dedis/kyber/, 2020.

[23] U. S. Department. Privileges and immunities, 2018. URL https://www.state.gov/ofm/accreditation/privilegesandimmunities/index.htm.

[24] L. C. C. Desmond, C. C. Yuan, T. C. Pheng, and R. S. Lee. Identifying unique devices through wireless fingerprinting. In *Proceedings of the first ACM conference on Wireless network security*, pages 46–55. ACM, 2008.

[25] DeterLab. Deterlab: Cyber-defense technology experimental research laboratory. https://www.isi.deterlab.net, 2016.

[26] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[27] R. Dubin, A. Dvir, O. Pele, and O. Hadar. I know what you saw last minute—encrypted http adaptive video streaming title classification. *IEEE Transactions on Information Forensics and Security*, 12(12):3039–3049, 2017.

[28] C. Dwork, A. Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[29] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 332–346. IEEE, 2012.

[30] J. Franklin, D. McCoy, P. Tabriz, V. Neagoe, J. V. Randwyk, and D. Sicker. Passive data link layer 802.11 wireless device driver fingerprinting. In *USENIX Security Symposium*, volume 3, pages 16–89, 2006.

[31] M. J. Freedman and R. Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 193–206. ACM, 2002.

[32] P. García, J. van de Graaf, A. Hevia, and A. Viola. Beating the birthday paradox in dining cryptographer networks. In *International Conference on Cryptology and Information Security in Latin America*, pages 179–198. Springer, 2014.

[33] P. Garcia, J. Van de Graaf, G. Montejano, D. Riesco, N. Debnath, and S. Bast. Storage optimization for non interactive dining cryptographers (nidc). In *Information Technology-New Generations (ITNG), 2015 12th International Conference on*, pages 55–60. IEEE, 2015.

[34] S. Goel, M. Robson, M. Polte, and E. Sirer. Herbivore: A scalable and efficient protocol for anonymous communication. Technical Report TR2003-1890, Cornell University, 2003.

[35] P. Golle and A. Juels. Dining cryptographers revisited. In *Eurocrypt*, 2004.

[36] X. Gong, N. Kiyavash, and N. Borisov. Fingerprinting websites using remote traffic analysis. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 684–686. ACM, 2010.

[37] J. Hall, M. Barbeau, and E. Kranakis. Enhancing intrusion detection in wireless networks using radio frequency fingerprinting. In *Communications, internet, and information technology*, pages 201–206, 2004.

[38] H.-C. Hsiao, T. H.-J. Kim, A. Perrig, A. Yamada, S. C. Nelson, M. Gruteser, and W. Meng. Lap: Lightweight anonymity and privacy. In *2012 IEEE Symposium on Security and Privacy*, pages 506–520. IEEE, 2012.

[39] J. Korhonen and Y. Wang. Effect of packet size on loss rate and delay in wireless links. In *IEEE Wireless Communications and Networking Conference, 2005*, volume 3, pages 1608–1613. IEEE, 2005.

[40] A. Krasnova, M. Neikes, and P. Schwabe. Footprint scheduling for dining-cryptographer networks. In *International Conference on Financial Cryptography and Data Security*, pages 385–402. Springer, 2016.

[41] A. Kwon, D. Lazar, S. Devadas, and B. Ford. Riffle: An efficient communication system with strong anonymity. In *PETS*, 2016.

[42] A. Kwon, H. Corrigan-Gibbs, S. Devadas, and B. Ford. Atom: Horizontally scaling strong anonymity. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 406–422. ACM, 2017.

[43] A. Kwon, D. Lu, and S. Devadas. {XRD}: Scalable messaging system with cryptographic privacy. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, pages 759–776, 2020.

[44] L. Lamport et al. Paxos made simple. *ACM Sigact News*, 32 (4):18–25, 2001.

[45] A. Langley. Pond, 2016. https://github.com/agl/pond.

[46] D. Lazar and N. Zeldovich. Alpenhorn: Bootstrapping secure communication without leaking metadata. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 571–586, 2016.

[47] D. Lazar, Y. Gilad, and N. Zeldovich. Karaoke: Distributed private messaging immune to passive traffic analysis. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 711–725, 2018.

[48] D. Lazar, Y. Gilad, and N. Zeldovich. Yodel: strong metadata security for voice calls. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 211–224, 2019.

[49] S. Le Blond, D. Choffnes, W. Zhou, P. Druschel, H. Ballani, and P. Francis. Towards efficient traffic-analysis resistant anonymity networks. *ACM SIGCOMM Computer Communication Review*, 43(4):303–314, 2013.

[50] S. Le Blond, D. Choffnes, W. Caldwell, P. Druschel, and N. Merritt. Herd: A scalable, traffic analysis resistant anonymity network for voip systems. *ACM SIGCOMM Computer Communication Review*, 45(4):639–652, 2015.

[51] S. Le Blond, A. Cuevas, J. R. Troncoso-Pastoriza, P. Jovanovic, B. Ford, and J.-P. Hubaux. On enforcing the digital immunity of a large humanitarian organization. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 424–440. IEEE, 2018.

[52] N. Mathewson and R. Dingledine. Practical traffic analysis: Extending and resisting statistical disclosure. In *International Workshop on Privacy Enhancing Technologies*, pages 17–34. Springer, 2004.

[53] R. Meier, D. Gugelmann, and L. Vanbever. itap: In-network traffic analysis prevention using software-defined networks. In *Proceedings of the Symposium on SDN Research*, pages 102–114, 2017.

[54] C. A. Neff. Verifiable mixing (shuffling) of ElGamal pairs. *VHTi Technical Document, VoteHere, Inc.*, 2003.

[55] L. Nguyen and R. Safavi-naini. Breaking and mending resilient mix-nets. In *PETS*, pages 66–80, 2003.

[56] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, pages 103–114. ACM, 2011.

[57] T. Peng, C. Leckie, and K. Ramamohanarao. Protection from distributed denial of service attacks using history-based ip filtering. In *IEEE International Conference on Communications, 2003. ICC'03.*, volume 1, pages 482–486. IEEE, 2003.

[58] B. Pfitzmann. Breaking an efficient anonymous channel. In *Advances in Cryptology-Eurocrypt 1995*, 1995.

[59] C. Phillips and S. Singh. CRAWDAD dataset pdx/vwave (v. 2007-09-14). Downloaded from http://crawdad.org/pdx/vwave/20070914/wlan_pcap, Sept. 2007. traceset: wlan_pcap.

[60] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis. The loopix anonymity system. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 1199–1216, 2017.

[61] M. A. Poletto and A. E. Dudfield. Architecture to thwart denial of service attacks, Feb. 2 2010. US Patent 7,657,934.

[62] PriFi. PriFi - Github. https://www.github.com/dedis/prifi, 2020.

[63] PriFi. PriFi Logs - Github. https://github.com/lbarman/prifi-experiments, 2020.

[64] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for web transactions. *ACM transactions on information and system security (TISSEC)*, 1(1):66–92, 1998.

[65] P. Rogaway. Nonce-based symmetric encryption. In *International Workshop on Fast Software Encryption*, pages 348–358, 2004.

[66] T. Ruffing, P. Moreno-Sanchez, and A. Kate. P2p mixing and unlinkable bitcoin transactions. In *NDSS*, 2017.

[67] J. Sankey and M. Wright. Dovetail: Stronger anonymity in next-generation internet routing. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 283–303. Springer, 2014.

[68] R. Schuster, V. Shmatikov, and E. Tromer. Beauty and the burst: Remote identification of encrypted video streams. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 1357–1374, 2017.

[69] S. Sengupta, H. Gupta, N. Ganguly, B. Mitra, P. De, and S. Chakraborty. CRAWDAD dataset iitkgp/apptraffic (v. 2015-11-26). Downloaded from http://crawdad.org/iitkgp/apptraffic/20151126/apptraffictraces, Nov. 2015. traceset: apptraffictraces.

[70] D. Shah et al. Gossip algorithms. *Foundations and Trends® in Networking*, 3(1):1–125, 2009.

[71] R. K. Sheshadri and D. Koutsonikolas. On packet loss rates in modern 802.11 networks. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2017.

[72] R. Skowyra, K. Bauer, V. Dedhia, and H. Okhravi. Have no phear: Networks without identifiers. In *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, pages 3–14, 2016.

[73] E. Syta, B. Peterson, D. I. Wolinsky, M. Fischer, and B. Ford. Deniable anonymous group authentication.

Technical Report YALEU/DCS/TR-1486, Department of Computer Science, Yale University, 2014. Available at http://cpsc.yale.edu/sites/default/files/files/TR1486.pdf.

[74] M. Tibouchi. Elligator squared: Uniform points on elliptic curves of prime order as uniform random strings. In *International Conference on Financial Cryptography and Data Security*, pages 139–156. Springer, 2014.

[75] N. Tyagi, Y. Gilad, D. Leung, M. Zaharia, and N. Zeldovich. Stadium: A distributed metadata-private messaging system. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 423–440, 2017.

[76] I. T. Union. ITU-T G.114 - Amendment 2: New Appendix III – Delay variation on unshared access lines, 2009. URL https://www.itu.int/rec/T-REC-G.114-200911-I!Amd2/en.

[77] J. Van Den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 137–152. ACM, 2015.

[78] VoIP-Info. VOIP QoS requirements. https://www.voip-info.org/wiki/view/QoS, 2017.

[79] T. Wang and I. Goldberg. Improved website fingerprinting on tor. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 201–212. ACM, 2013.

[80] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg. Effective attacks and provable defenses for website fingerprinting. In *USENIX Security Symposium*, pages 143–157, 2014.

[81] A. M. White, A. R. Matthews, K. Z. Snow, and F. Monrose. Phonotactic reconstruction of encrypted voip conversations: Hookt on fon-iks. In *2011 IEEE Symposium on Security and Privacy*, pages 3–18. IEEE, 2011.

[82] P. Winter, T. Pulls, and J. Fuss. Scramblesuit: A polymorphic network protocol to circumvent censorship. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 213–224. ACM, 2013.

[83] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Dissent in numbers: Making strong anonymity scale. In *OSDI*, 2012.

[84] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Scalable anonymous group communication in the anytrust model. In *EuroSec*, 2012.

[85] D. I. Wolinsky, E. Syta, and B. Ford. Hang with your buddies to resist intersection attacks. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1153–1166. ACM, 2013.

[86] C. V. Wright, S. E. Coull, and F. Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *NDSS*, volume 9. Citeseer, 2009.

[87] Q. Xu, R. Zheng, W. Saad, and Z. Han. Device fingerprinting in wireless networks: Challenges and opportunities. *IEEE Communications Surveys & Tutorials*, 18(1):94–104, 2016.

[88] T. Zhu, D. Feng, Y. Hua, F. Wang, Q. Shi, and J. Liu. Mic: An efficient anonymous communication system in data center networks. In *2016 45th International Conference on Parallel Processing (ICPP)*, pages 11–20. IEEE, 2016.

A Proofs of Properties

A.1 Setup

Property 1. A shared secret r_{ij} between an honest client C_i and an honest guard S_j is known only to C_i and S_j.

Proof. The shared secrets r_{ij} are derived using an authenticated Diffie-Hellman protocol; due to the hardness of the DDH assumption in $\mathbb{G}$ (which implies hardness of DLP), $\mathcal{A}$ is unable to recompute r_{ij} without the private key of C_i or S_j. □

Property 2. Let C_0, C_1 be two honest clients who ran `Setup` without aborting, and $\alpha(0)$,$\alpha(1)$ the position their respective shuffled key in π. Then, the adversary $\mathcal{A}$ has negligible advantage in guessing $b \in [0,1]$ with significant advantage such that the mappings client $\rightarrow$ position $(b \rightarrow \alpha(0),(1-b) \rightarrow \alpha(1))$ are in π.

Proof. If `Setup` terminates without aborting for an honest client C_i, then, as a consequence of the checks done by C_i, it means that: (1) the verifiable shuffle completed correctly, (2) π is signed by every guard in G, (3) there are at least $K=2$ clients in π, and (4) their own pseudonym is included in π.

Without loss of generality, let us assume that S_1 is the honest server in G. Since π is signed by all $S_j \in G$, then S_1 participated in Neff's Verifiable Shuffle [54] and signed the output π.

Since G contains at least 2 honest users, and both C_1 and C_2 ran `Setup` without aborting, it means that their two pseudonyms are included in π. Since both are honest, their ephemeral private keys p_1,p_2 are unknown to $\mathcal{A}$.

Therefore, the Neff shuffle ran with at least 2 keys as input, which are the keys of the honest clients (due to check 4), and at least one honest server shuffled the keys. Let $\alpha(0)$,$\alpha(1)$ be the position of C_1 and C_2's respective shuffled keys in π. Without p_1, $\mathcal{A}$ is unable to differentiate between $\tilde{P}_{\alpha(0)}$ and $\tilde{P}_{\alpha(1)}$, and can do no better than random guessing. The same argument can be made for C_2. □

A.2 Anonymize

Property 3. After a run of `Anonymize`, let C_{i_1}, C_{i_2} be two honest clients, $k_1 = \alpha(i_1)$,$k_2 = \alpha(i_2)$ be the time slots in which they communicated, and let m_{k_1}, m_{k_2} be the anonymous upstream messages for those slots. Then, $\mathcal{A}$ has negligible advantage in guessing $b \in [1,2]$ such that m_{k_b} is the message sent by i_1.

Proof. $\mathcal{A}$ tries to distinguish between $(C_{i_1} \rightarrow m_{k_1}, C_{i_2} \rightarrow m_{k_2})$ and $(C_{i_1} \rightarrow m_{k_2}, C_{i_2} \rightarrow m_{k_1})$. For simplicity, we define the following equivalent game: for a slot $k \in \{k_1, k_2\}$, $\mathcal{A}$ guesses $b \in [1,2]$ such that C_{i_b} is the anonymous sender of the message m_k.

To ease notation, let C_{i_1} be C_1, and C_{i_2} be C_2. Without loss of generality, let S_1 be a honest guard, and let C_1 be the anonymous sender. Then, on slot k_1, we have

$$\begin{aligned} c_1 &= \bigoplus_{j=1}^{m} \texttt{PRG}(r_{1,j}) \oplus m_1 \\ c_2 &= \bigoplus_{j=1}^{m} \texttt{PRG}(r_{2,j}) \end{aligned} \tag{1}$$

Since S_1,C_1 and C_2 are honest, the values $\texttt{PRG}(r_{1,1})$ and $\texttt{PRG}(r_{2,1})$ are unknown to $\mathcal{A}$. We isolate the contribution of S_1:

$$\begin{aligned} c_1 &= \bigoplus_{j=2}^{m} \texttt{PRG}(r_{1,j}) \oplus \texttt{PRG}(r_{1,1}) \oplus m_1 \\ c_2 &= \bigoplus_{j=2}^{m} \texttt{PRG}(r_{2,j}) \oplus \texttt{PRG}(r_{2,1}) \end{aligned} \tag{2}$$

All values have the same length. Both values $\texttt{PRG}(r_{1,1})$ and $\texttt{PRG}(r_{2,1})$ are the output of a PRG and are indistinguishable from a random string in the random oracle model. Therefore, $\texttt{PRG}(r_{1,1}) \oplus m_1$ is also indistinguishable from a random string, and hence so are both strings c_1 and c_2. Hence, since $r_{1,1}$ and $r_{1,2}$ are unknown to $\mathcal{A}$, if PRG is correctly instantiated, $\mathcal{A}$ sees two random strings and can only guess b with probability 1/2. □

A.3 Disruption-Protection

Property 4. The anonymity of any honest client is unaffected by the information $\underline{\texttt{PRG}(r_{ij})_l}$ made public in step 2 of `Blame` (Protocol 4).

Proof. The values $\texttt{PRG}(r_{ij})_l$ for slot k de-anonymize precisely the l^{th}-bit of the slot k, by revealing the composition of the messages $\underline{c_i}$ and $\underline{s_j}$ at position l.

Let C_i be the (honest) owner of slot k. Only C_i can generate the proof of knowledge $\text{PoK}_{k,l}(\tilde{p}_k : \tilde{p}_k = \log\ \tilde{P}_k)$, and honest clients verify the PoK before revealing any information, hence honest clients do not reveal information about a slot without the owner's consent.

In slot k, all non-sender, honest clients transmitted a 0 on bit l, since $k \neq \alpha(i)$ and the protocol forces them to transmit $\underline{0}$. Additionally, the slot owner C_i only sends a PoK for l such that $(m_k)_l = 0$. Therefore, on slot k, at position l, all honest clients transmitted a 0. The values revealed match the $(c_i)_l$ values previously sent, and the $(c_i)_l$ are composed solely the output of the PRGs seeded by the secrets, which are indistinguishable from a random string in the random oracle model. Finally, the knowledge of one given bit reveals nothing about other parts of the PRG output. □

Property 5. The anonymity of any honest client is unaffected by the information r_{ij} made public in step 5 of `Blame` (Protocol 4).

Proof. By revealing r_{ij}, a pair of client-guard (C_i, S_j) completely reveal their contributions to the DC-net. However, a honest client and a honest guard never contradict each other, as the protocol ensures that their values $\texttt{PRG}(r_{ij})_l$ are correctly computed from the common shared secret r_{ij}. Since in step 5, a honest client C_i only reveals r_{ij} when seeing (1) a signed message from a guard, (2) for which there is a contradiction, he never reveals r_{ij} for an honest guard S_j. The same argument can be make for S_j about any honest client.

Therefore, those values are revealed only when C_i or S_j is malicious (or both); in this case, $\mathcal{A}$ already had knowledge of r_{ij}. □

Property 6. Let C_i be the owner of a slot k, and let C_d, $d \neq i$ be another client (or guard). If C_d sends an arbitrary value c_i' instead of the value c_i (or s_j) as specified in the protocol, then C_d is identified as the disruptor and is excluded from subsequent communications.

Proof. We show the following slightly weaker statement: in the case of multiple disruptors, the protocol excludes *a* disruptor C_d. We argue that since all are under the control of $\mathcal{A}$, *which* disruptor gets excluded is irrelevant. By repeating this argument, eventually, all disruptors gets excluded.

The proof describes a client disruptor; the argument for a guard is analogous.

Let c_i' be the value sent instead of c_i. Then, we can write $c_i' = c_i \oplus q$, and we observe that $m_k = m_i \oplus q$. Let $|m_i|$ be the length of m_i, and z be the number of 0 bits in m_i. The adversary disrupts stealthily (*i.e.,* is not detected) if no 0 in m_i is flipped to a 1 (in this case, the honest client cannot start a `Blame` without de-anonymizing himself).

If m_i is unknown to $\mathcal{A}$ and is uniformly distributed in $\{0,1\}^{|m_i|}$, the probability that no 0-bit is flipped follows $(\frac{1}{2})^z$, with $\mathbb{E}[z] = \frac{|m_i|}{2}$.

If, however, $\mathcal{A}$ has knowledge of some distribution of m_i (*e.g.,* because perhaps, after a long period of silence, the first message of a client is often an unencrypted DNS request), then $\mathcal{A}$ has a much better chance of targeting the 1 bits. Fortunately, the clients are free to compose m_i how they want, *e.g.,* by starting with several 0's (0...0||DNS||0...0), so we expect that in practice the adversary should have little information about the distribution of each bit in m_i. We note that clients can notice disruption and vary how they compose m_i, or even send a "trap-

message" of all 0's (which allows to catch a disruptor with probability 1) should they notice significant disruption. □

Property 7. An honest entity is never identified as a disruptor.

Proof. Trivially, honest clients follow the protocol. Each step of the `Blame` protocol consists of revealing some already-performed step of `Setup` and `Anonymize` protocol; by definition, honest entities performed those correctly. □

A.4 Equivocation-Protection

Property 8. The anonymity of any honest client is unaffected by the extra information κ_i sent in `DCNet-Gen-Client`.

Proof. For a slot k which does not go through a `Blame` procedure, the adversary knows κ_i for all clients, h_i, and all $\mathtt{PRG}(r_{ij})$ for a malicious C_i, or a malicious S_j, but not between a honest pair (C_i,S_j). Without loss of generality, assume C_1,C_2,S_1 are the honest clients and guard. Then, $\kappa_1 = F_1(\gamma) + c_1 + c_2$, where c_1 is a constant known to $\mathcal{A}$, and $c_2 = F_2(h_i) \cdot F_3(H(\mathtt{PRG}(r_{1,1})))$ is unknown to $\mathcal{A}$ and distributed uniformly at random in $\mathbb{G}$ due to F_2. $\mathcal{A}$ is unable to distinguish between $F_1(\gamma)+c_2$ and c_2, both being random values uniformly distributed in $\mathbb{G}$, and hence cannot distinguish between the k_i of a honest sender and of a honest non-sender.

If the slot k goes through a `Blame` procedure, then in addition to the previous knowledge, for any i,j, at most one bit of $\mathtt{PRG}(r_{ij})$ is revealed in step 2 of `Blame`; this value is used in the computation of κ_i and σ_j, but this single bit does not reveal information about $H(\mathtt{PRG}(r_{ij}))$.

A private value $\mathtt{PRG}(r_{ij})$ (between a pair of honest client/guard C_i and S_j) cannot be recovered from κ_i by $\mathcal{A}$ due to the use of the cryptographic hash function H. □

Property 9. The anonymity of any honest client is unaffected by the extra information σ_j sent in `DCNet-Gen-Guard`.

Proof. The σ_j are independent from the communicated content and the slot owner, and reveal nothing about these.

As discussed in Proof of Property 8, due to the use of the cryptographic hash function H, the σ_j do not allow to recover any $\mathtt{PRG}(r_{ij})$, nor does revealing 1 bit of $\mathtt{PRG}(r_{ij})$ allow to recover $H(\mathtt{PRG}(r_{ij}))$. □

Property 10. The anonymity of any honest client is unaffected by the extra information $\underline{Q_i}$,PoK sent in step 7 of `Blame`.

Proof. The Q_i are independent from the communicated content and the slot owner, and reveal nothing about these.

Each value Q_i in $\underline{Q_i}$ has the form $Q = P \cdot F_2(H(\mathtt{PRG}(r_{ij})))$, where P is a base point of $\mathbb{G}$, in which the DLP problem is hard. Without loss of generality, assume C_1,C_2,S_1 are the honest clients and guard. Then, $F_2(H(\mathtt{PRG}(r_{11})))$ and $F_2(H(\mathtt{PRG}(r_{21})))$ are unknown to $\mathcal{A}$, and the Q_i do not help in computing them due to the assumption on $\mathbb{G}$. Therefore, both C_1 and C_2 send a value (1) unknown to $\mathcal{A}$ and (2) uniformly distributed in $\mathbb{G}$ due to F_2, hence $\mathcal{A}$ has no advantage in telling them apart.

Due to the hardness of the DLP in $\mathbb{G}$, the values Q_i give no information about κ_i.

Finally, the zero-knowledge proof of knowledge PoK trivially reveals no information. □

Property 11. The anonymity of any honest client is unaffected by the information r_{ij} made public in step 9 of `Blame`.

Proof. This proof is a direct copy of the proof of Property 5. □

Property 12. If $\exists i,j$ two honest clients who received $\underline{d_i} \neq \underline{d_j}$ on round k, then neither the relay, nor $\mathcal{A}$ can decrypt m_k for any subsequent round $k' > k$.

Proof. Without loss of generality, let C_1,C_2 be two honest clients who received $\underline{d_1} \neq \underline{d_2}$ at the end of round k. Then, with overwhelming probability, $h_1 \neq h_2$ due to the use of the cryptographic hash function, and it follows that $F_2(h_1) \neq F_2(h_2)$ with high probability.

To simplify the proof, assume h_1 is the "correct" history. Therefore, the correct value for κ_2 (if C_2 had received the correct downstream message $\underline{d_1}$) would be

$$\kappa_2' = F_2(h_1) \cdot \sum_{j=1}^{m} F_3(H(\mathtt{PRG}(r_{ij})))$$

Without loss of generality, assume S_1 is the honest guard; we rewrite the above equation, isolating $r_{2,1}$:

$$\kappa_2' = F_2(h_1) \cdot F_3(H(\mathtt{PRG}(r_{2,1}))) + F_2(h_1) \cdot \sum_{j=2}^{m} F_3(H(\mathtt{PRG}(r_{ij})))$$

But since C_2 and S_1 are honest, $r_{2,1}$ is unknown to $\mathcal{A}$ and so is the scalar $F_3(H(\mathtt{PRG}(r_{2,1})))$, and therefore $\mathcal{A}$ is unable to recompute the decryption key γ for any round $k' > k$.

We note that the proof does not requires C_1 nor anyone to have a "correct" history, simply $h_1 \neq h_2$ for two honest clients.

Additionally, while this property ensures that an equivocation attack never leads to de-anonymizing the client through their anonymous output, incidentally, it gives honest clients a proof $\mathcal{S}_{\hat{p}_r}(h_r)$, with $h_r \neq h_i$, that the relay did not behave correctly; administrative can then be taken against the relay. □

Property 13. If a client C_i sends an arbitrary value κ_i' instead of the value κ_i as specified in the protocol, then C_i is identified as the disruptor and is excluded from subsequent communications.

Proof. The argument is analogous to the proof of Property 6: if the honest client whose slot was disrupted can start a blame, then the `Blame` protocol ultimately excludes one disruptor. □

Property 14. An honest relay is never accused of an equivocation attack.

Proof. Since the since the downstream messages $\underline{d}$ are signed by the honest relay, the adversary is unable to forge a message sent by the honest relay. Since an honest relay never equivocates, a malicious client attempting to frame the relay is equivalent to a malicious client sending wrong $\underline{c_i}$,$\underline{\kappa_i}$ values.

When this happens during an honest client slot, it will eventually start a `Blame` procedure. The `Blame` procedure verifies the correctness of the $\underline{c_i}$,$\underline{\kappa_i}$ values since, for a correct h_i, the $\underline{c_i}$,$\underline{\kappa_i}$ of all parties but the honest sender are fully determined by the shared secrets r_{ij}. The honest sender trivially sends the correct values.

When this happens during a malicious client slot, the client may wrongly compute κ_i and still pass the `Blame` due to the first "Or" of the PoK. In this case, no party is identified as a disruptor.

□

Property 15. An honest client is never excluded as a disruptor.

Proof. The end of `Blame` (Protocol 4) describes more precisely what it means for a client C_i to be "excluded as a disruptor": the relay broadcasts a new group G' and roster T' without the public key of C_i. The process for a server is analogous, except that $T' = T$.

The proof is done for an honest client, and is analogous for an honest server.

Trivially, an honest client always produces correct values. During a `Blame` procedure, every message is signed. The honest client first sends $\underline{\texttt{PRG}(r_{ij})}$ for slot k, which matches what he sends in round k. Then, if he is not in contradiction with some server, the honest client stops taking part in `Blame` (and is not excluded); let us assume he is in contradiction with S_j. Then, upon reception of the signed message from S_j, he reveals r_{ij} along with a proof of correctness.

To exclude a client (by broadcasting a new group G' and roster T), the relay also has to reveal all inputs of `Blame` for accountability. To exclude an honest client without blatantly cheating, the relay would have to forge one signature, which contradicts our adversarial model.

In practice, we note that the malicious relay could update G and T without justification; however, this is analogous to a simple denial-of-service and contradicts the threat model. In practice, this would prompt clients to take administrative actions against the relay. □

B Additional Evaluation Results

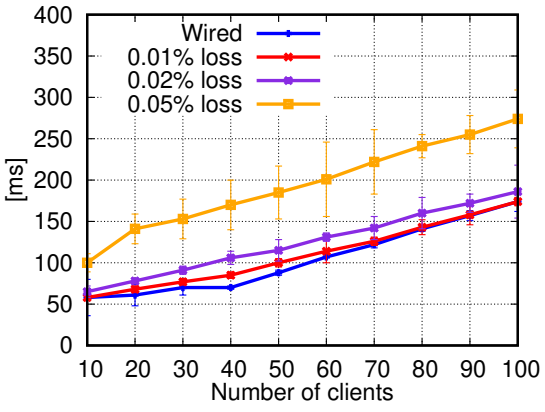


Fig. B.1. Latencies when varying the loss rate.

CPU/Memory. We briefly evaluate the CPU usage on the relay during an Anonymize round; our model estimates less than 10% of average usage for 100 clients on commodity hardware (Figure B.7). We tested the memory and CPU usage on an Android Nexus 5X device; the device on which the measurements are done is doing light web browsing activities through PriFi. During Anonymize, the mean CPU usage is light (below 5%), and the memory usage is moderate (stable at 50 Mb; for comparison, Telegram uses 150 Mb). The energy consumption fluctuates between "Light" and "Medium" (estimated with Android Studio 3.2 Beta) and is comparable to a VoIP call.

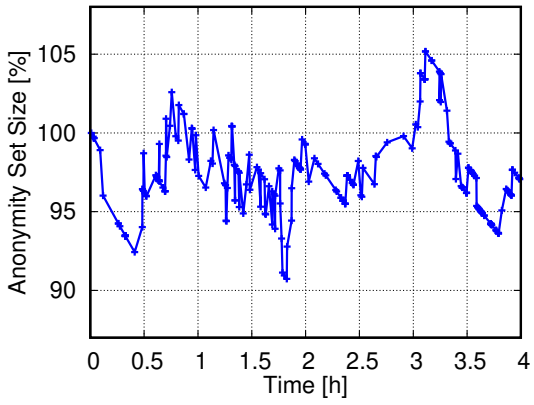


Fig. B.2. Size of the anonymity set in the café scenario. This shows among how many users a PriFi client is anonymous.

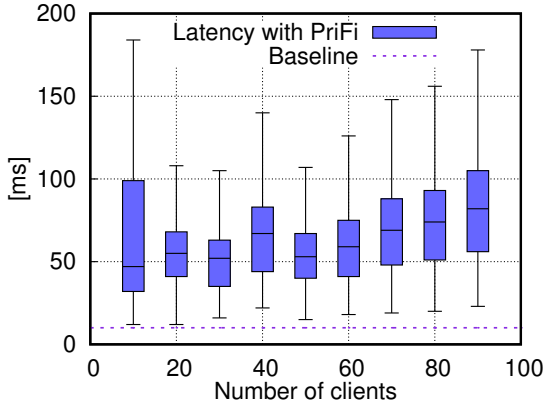


Fig. B.3. 5% of users performing various HTTP(S) requests and file downloads.

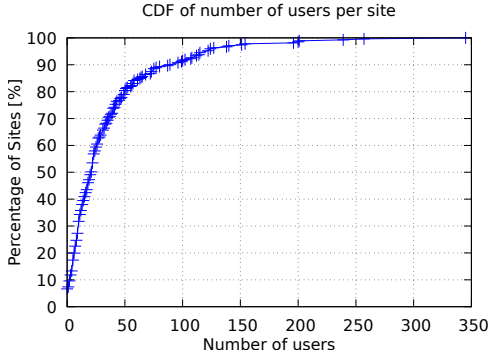


Fig. B.4. Distribution of users per ICRC site. 90% of sites have less than 100 users.

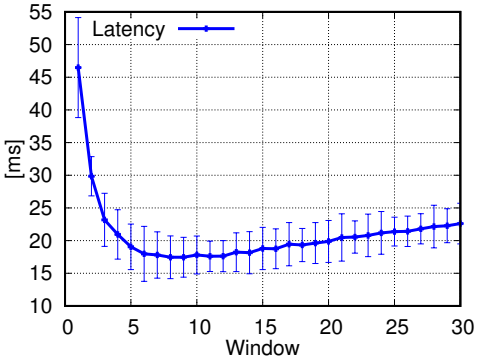


Fig. B.5. Effect of pipelining on latency. A window $W=7$ divides the latency by 2.25 in comparison to the naïve $W=1$ approach.

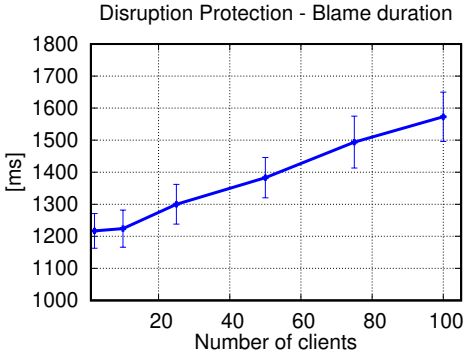


Fig. B.6. Duration of the blame procedure used to exclude a malicious client performing a disruption attack. Dissent's non-probabilistic version needs "minutes to hours" to exclude a disruptor [83] (with probability 1, unlike our protocol which has probability 1 to detect but only $1/2$ to exclude).

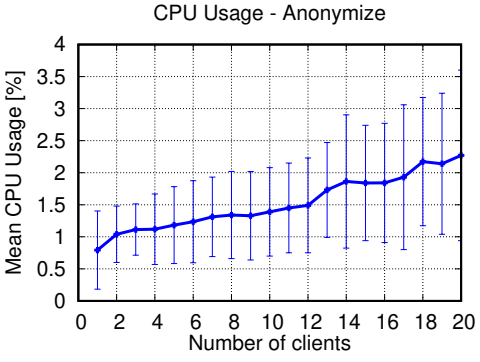


Fig. B.7. CPU usage on the relay during Anonymize, averaged over 10 minutes. The client are real Android and iOS devices (hence the x axis stopping at 20). The relay's hardware is a commodity server with a 3GHz Xeon Dual Core and 2GB of RAM. If we extrapolate the linear tendency, we estimate the mean CPU usage to be below 10% with 100 clients in this setup.

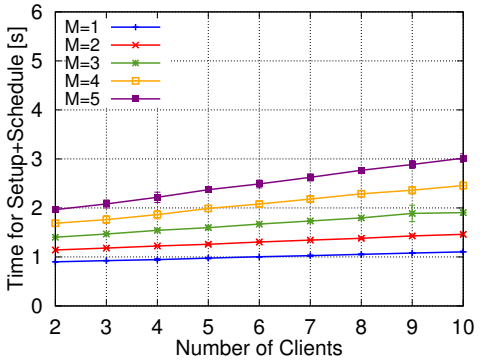


Fig. B.8. Duration of `Setup`. This corresponds to the downtime in case of client churn.