Performance Analysis & Enhancements for PriFi

Ludovic Barman
EPFL

Advisor: Italo Dacosta
LCA1, EPFL

Advisor: Prof. Bryan Ford
DeDiS, EPFL

Advisor: Prof. Jean-Pierre Hubaux
LCA1, EPFL

In an effort to thwart today's mass surveillance, and to provide better alternatives to entities who value anonymity, we propose PriFi, a novel anonymous communication network that leverages on a specific (but common) network setup to provide both *traffic-analysis resistance* against a global passive adversary, and *low-latency communication*. PriFi relies on a set of public servers *off the latency path*, and does not add an extra hop between the users and the Internet. The result is a trustless VPN service that anonymizes any kind of traffic, even delay-sensitive ones, and provides local, strong anonymity.

1. Introduction

In today's world of mass internet surveillance [1, 5, 21] and control of information [22, 25, 31], an increasing number of people feel unsafe while communicating on the Internet [14, 19]. Freedom of speech, a key to modern and open societies, is being threatened by the constant recording of online human interactions by states and other powerful entities. Encryption solves only partially the problem, as it does not hide the entities communicating. The remaining contextual information, the *metadata*, allows the aforementioned entities to spy and track people on a scale never experienced before; a dramatic situation, especially in oppressive regimes. One of the ways to reclaim freedom of speech is via *anonymous communication networks* that allow people to share information while being indistinguishable from their peers, and hence untraceable by an eavesdropping entity.

The most widespread anonymous network, Tor [30], was not designed to protect against global surveillance [33]; worse, it is now established that state-level adversaries can and do *de-anonymize* Tor users [24, 29]. Protecting against traffic-analysis is inherently *costly*, either in terms of *latency* or *scalability*. In our work, we build a protocol that leverages on a specific network infrastructure to keep the overhead tolerable. Based upon Dining Cryptographer's networks, or *DC-nets* [6], our protocol provides *provable anonymity* against a global passive adversary.

Our contribution, PriFi [1], is an application-agnostic protocol with provable anonymity against a global passive adversary. It works like a trustless VPN, and creates an anonymous communication network on a (W)LAN[2], anonymizing user's traffic. A key challenge is to keep the system as *low-latency* as possible, to support classes of traffic like VoIP or video conference. To achieve this, PriFi relies on the notion of *latency-path*, i.e. the path taken by packets towards and from the Internet, and does *not* add an extra hop that increases delays. Helper servers (called *trustees*) are kept *off the latency path*, and only add latency during setup.

[1] *Pri*vacy-protecting Wi-*Fi*.

[2] We use the notation *(W)LAN* to denote both Wireless and wired LAN networks. The mention W is kept (instead of LAN, simply) due to the extra optimizations possible when using a WLAN network.

This paper is a follow-up to the previous report on PriFi [3], and focuses on:

1. Mobile aspects of PriFi: the feasibility and cost of using PriFi with mobile nodes (e.g. smartphones)
2. Engineering aspects of PriFi: the architectural changes made in the prototype to support multicast

This paper is organized as follows: Section 2 presents a fundamental trade-off between *traffic-analysis resistance*, *low-latency* and *bandwidth efficiency*. In Section 3, we briefly summarize our protocol PriFi and its particularities; we study the mobile aspects of PriFi in Section 4, and discuss engineering problems in Section 5. We conclude in Section 6 by proposing directions for future work.

2. Background

Generally speaking, anonymous communication networks cannot have at the same time *traffic-analysis resistance*, *low-latency* (e.g. low enough to be compatible with VoIP, or other latency-sensitive applications), and *bandwidth efficiency* (the absence of significant padding or chaff traffic[3]).

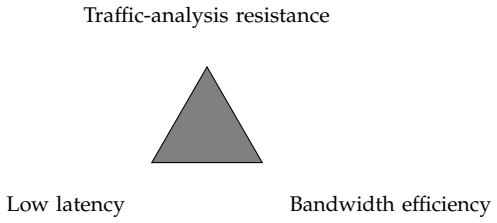


Figure 1: Impossibility triangle - pick only two out of three properties.

A network that wants *traffic-analysis resistance* needs its users to have identical traffic time-series; **(1)** without chaff traffic (with *bandwidth efficiency*), every client needs to synchronize and wait arbitrarily long on the others, thus the network probably does not have *low-latency*. **(2)** Without depending on the sending rate of other users, which is a necessary condition for *low-latency*, the time-series of the traffic between users would look different without *chaff traffic* (i.e. which makes the network not *bandwith efficient*).

A network that does not add chaff traffic (e.g. matching our definition of *bandwidth efficiency*) and does not wait on the slowest user (e.g. with *low latency*) does not provide traffic-analysis resistance, as trivially the fast users behave differently than the slow ones.

[3] Chaff traffic is dummy traffic; here, it is used to send packets even where there is no information to send.

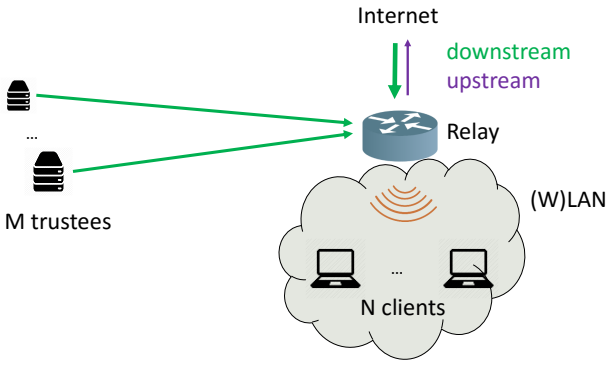


Figure 2: System setup

A similar result (with the addition of a 4-th property, *Resistance to catastrophic DoS attack*) have been presented in [2].

PriFi. Our protocol PriFi leverages on DC-nets [6], and has *traffic-analysis resistance*. To be usable as an application-agnostic VPN, we need *low-latency*, for supporting delay-sensitive traffic. As a result, our protocol will not be *bandwidth efficient*: we will need significant chaff traffic, but we show that the overhead is tolerable in our setup.

3. System Design

3.1. Model

We consider a set $\mathbf{U}$ of N users who want to anonymize their communications, and a set $\mathbf{T}$ of M servers (called *trustees*) whose task is to assist the clients in the anonymization process. Users are connected to the Internet through a relay, or router R, which processes normal traffic in addition to running our protocol. This setup is depicted in Figure 2.

Trustees are public servers located anywhere in the world; their link to the relay R is presumably *high-latency*, in the order of 100 ms. The clients and the relay on the contrary are *close-by*, and are connected with *low-latency links*, typically through a LAN or a WLAN. We select this setup to preserve *low-latency*: the anonymized traffic does *not* go through the trustees, and no additional hop is added to the path from the clients to the Internet.

Users, trustees and the relay will jointly run a protocol called PriFi. From the point of view of the users, it will behave like a trustless VPN, receiving data from and sending data to the applications running on the user's computer. The relay will act as the other end of the VPN, sending data to the Internet, or to the other users. Our protocol anonymizes the traffic between the client, and does not require the relay to be honest.

3.2. Attacker model

We define a *passive* attacker A that **(1)** can eavesdrop on the network and **(2)** controls any of the aforementioned entities: clients, servers, or relays. For simplicity, we define the set of entities controlled by the adversary as static, i.e. an entity is either controlled by A, or it will never be. As usual, we do not try to give any guarantees to malicious or corrupt parties.

Clients. We define $\mathbf{U}_H \subseteq \mathbf{U}$ the set of honest users, i.e. users not controlled by A. We restrict our scenario to the cases where $\|\mathbf{U}_H\| \geq 2$, since otherwise there is a unique honest client whose communication is trivially identifiable by A.

Trustees. The trustees are considered in the anytrust model [38]: they are all considered to be available at all time, and at least one of them is considered to be honest. Clients do not need to know *which one* is honest, and are secure as long as *any one* is.

Active attacks and denial of service. In this project, active attacks are not considered (yet). In the long term vision, PriFi needs to handle *active insider attacks* from malicious nodes; the feasibility of protecting against *active outsider attacks* has not been established yet.

3.3. Protocols

We split the protocol in two parts: the *downstream data* (i.e. the data from the Internet to one of the clients) is simply broadcasted from the relay to every client. The *upstream data* (i.e. the data from one of the clients towards the relay) goes through our anonymization protocol Anonymize, collectively run between all nodes, and the result of this protocol is outputted by the relay towards the Internet (or the clients, depending on the recipient of the packets).

Setup protocol. During this protocol, the users contact the relay, who broadcasts a list of trustees and their public keys. Clients then agree on the set of trustees and verify the authenticity of the public keys[4]. Then, clients and trustees exchange or derive the following secrets: each trustee $t \in \mathbf{T}$ and each user $u \in \mathbf{U}$ derives a secret $s_{u,t}$; the secret is then used in combination with a pseudo-random number generator (PRNG) to obtain a stream of pseudo-random bits, from which users and trustees will compute their ciphertexts.

Anonymize protocol. The anonymization protocol works in time slots, where clients send the *client ciphers* C_{u_i} to the relay, and so does the trustees with the *trustee ciphers* C_{t_i}. The time-slot ends when the relay collected exactly one cipher from every client and trustee.

Formally speaking, the anonymization protocol is a distributed protocol that outputs one message m per time-slot T, given the users' and the trustees' ciphers $C_u(T)$ and $C_t(T)$ for this time-slot.

$$m(T) \leftarrow \text{Anonymize}(C_{u_1}(T), \cdots, C_{t_1}(T), \cdots)$$

Resync protocol. So far, DC-nets have been described using static sets of clients and trustees. While trustees can be in fact considered to be always available (the extension to error-prone trustee is almost similar to the case of error-prone clients), clients can fail, hang, and purposely disconnect; those actions *break the current round*, that is, for this current timeslot T, Anonymize$(\cdots) \rightarrow \emptyset$. It can be easily seen that there is no easy fix to recover the lost data $m(T)$ for this round, as it is XOR'ed with a pseudo-random secret that is *not* cancelled out in the final XOR.

[4]The distribution of identities and public keys, and the prevention of a man-in-the-middle attack at this level, is out-of-scope of this short project. We assume all entities received the correct public keys.

When this happens, client needs to redo the Setup protocol where all the clients are discovered, and the keys are exchanged. During this time, the DC-net cannot process upstream data.

Schedule protocol. Once the DC-net is ready, and clients are ready to communicate, one last question must be answered: which client gets to transmit a message during which time slot. Perhaps the simplest way would be to allow clients to try, and retransmit if a collision is detected (i.e. if $m(T)$ is not the client's message); however, this re-introduces unneeded collision detection (on top of the network protocol used).

A more efficient way is to *schedule* which client gets which time-slot. It is critical that this mapping is secret, as otherwise it completely removes all the anonymity properties from the DC-net. For that purpose, we use an oblivious permutation protocol, where all the trustees help in permuting a set of public keys chosen by the clients, in a way that *no one* knows the full final permutation if at least one of the trustees is honest. We use the Neff-Shuffle [28], adapted to elliptic curves, which produces oblivious verifiable shuffle. The output is a set of different public keys; clients are only able to recognize their own public key in the set, other keys look unrelated to all parties without the associated private key.

This is a very basic version of a scheduling protocol, as it simply assigns one fixed-size slot per client. Several extensions could be created, including client negotiating for a certain number of slots, or variable-length slots, thus distributing the available bandwidth more efficiently.

4. Mobile aspects

We consider a corporate network running PriFi, which anonymizes the traffic from the employees. In some companies, employees are likely to have not only a corporate computer, but also a corporate smartphone; if the smartphones are not the property of the company, it is still reasonable to consider that employees use their own smartphone to handle the company's emails and to browse the Internet on work-related topics. Hence, we argue that in most cases, the traffic originating from smartphones is as sensitive and personal as the traffic originating from desktops. Following that idea, it would be interesting to have a mobile PriFi app (conceptually identical to the PriFi software run on desktops), anonymizing the mobile phone's traffic, perhaps in the same anonymity set as the desktops. In this section, we explore the feasibility of this approach.

4.1. Cellphones energy consumption

Modern smartphones now have computing capabilities on par with desktops computers, and could theoretically run PriFi without problems. The fundamental difference is that smartphones are *resource-limited*. While on desktop PriFi could freely send continuous cover traffic, ignoring the modest added CPU cost and the more significant transmission cost, mobile nodes have to cope with a limited battery.

Radio communication is one of the most significant aspect (controlled by applications[5]) regarding energy consumption [35]. Using the radio interface is expensive : sending and receiving messages prevents the radio interface from going into sleep mode, where is consumes significantly less energy. Google's guidelines [34] suggest that reducing the usage of the network interface is of capital importance for battery life, and that applications from different vendors should *collaborate* to send messages only when the interface is already up, and try not waking the interface from sleep.

Unfortunately, PriFi by definition goes *against* those guidelines: it forces *all* devices to send *frequently* (dozens to hundreds of messages per second as cover traffic), a *large* amount of data (the chaff traffic needs to allow other devices to communicate with a high enough bitrate). In PriFi, the following properties always hold:

1. **Equivalent data transmission**. Every device of one anonymity set sends the same amount of data (either chaff traffic, or actual information) on the network.
2. **Synchronicity**. Every device of one anonymity set sends the same amount of messages in the same time-slots.

This means that a *single* device that wants to transmit information forces *every other device* to send chaff traffic at the same time and rate. This scenario is clearly a terrible for battery life, as every device that could have been idle pays the price to allow some other device to communicate.

Hence, the feasability of the approach in the mobile world needs to be asserted. While several experiments have been run with desktops computers [3], and have confirmed the practicality of the approach, none so far have precisely targeted mobile phones.

4.2. PriFi-Sim

The preliminary analysis of the mobile scenario showed a lack of metric on the cost of anonymity: the argument "bandwidth is high enough in LAN" used previously is not satisfactory anymore, and we need a predictor for the cost of anonymizing the traffic of a set of devices. This cost will depend on the number of clients, the parameters of PriFi (e.g. the round scheduling algorithm), and more importantly, on the traffic these clients want to anonymize.

To answer those questions, we built a Go program called PriFi-Sim [4], a simulator for PriFi. The objectives of PriFi-Sim are twofold: **(1)** to evalutate the cost of running PriFi in a specific setup, and **(2)** to allow quick testing of policies (e.g. schedule algorithm) before implementing them in PriFi.

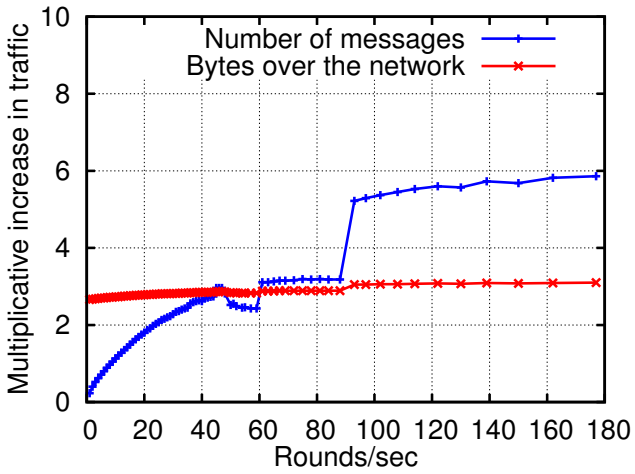
PriFi-Sim takes as input some packet traces in a standard format (.pcap[6]), along with a mapping of identities [7]. Then, given some parameters for PriFi such as the policy for round scheduling, the number of rounds per second, and if the number of round per second is fixed or variable, it proceeds to emulates PriFi by feeding the inputted traces as payload in the PriFi packets. In addition, it takes into account the synchronicity between the clients, as well as the lock-step between the clients and the relay.

The output is a new set of traces (also in .pcap format) corresponding to the PriFi packets. The payload of these packets will be the real traffic given as input. Given the constraints on PriFi packets, this real traffic will experience extra latency; additionally, nodes will send extra traffic, increasing the number of messages and the amount of

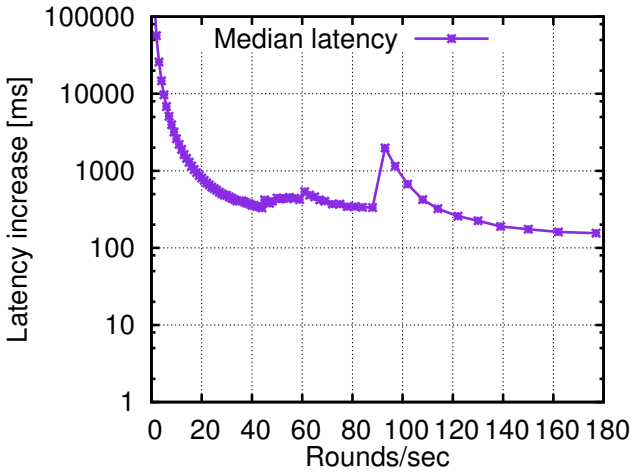
[5] Other aspects such as the brightness of the screen play an important role, but are defined by the user.

[6] Data file created by Wireshark (formerly Ethereal), a free program used for network analysis.

[7] In those traces, we select a subset of IPs, and declare them to be the clients of a PriFi protocol. Their traffic will be anonymized, and other IP's traffic will be left untouched.



(a) Multiplicative increase in number of messages and bytes sent



(b) Added latency

Figure 3: Increase in number of messages, bytes sent, and extra latency after running the dataset through PriFi-Sim. Mean over all PriFi nodes.

raw data sent over the network.

In a second phase, PriFi-Sim provides analytics both *per-node* and *globally*. It shows the differences in the number of messages sent, the total number of bytes sent, and the latency added to the real traffic due to bucketing.

Experiment setup. We run PriFi-Sim with the dataset *Dataset of wireless LAN traffic around Portland, Oregon using a commercial sniffer VWave* [10], provided by CRAWDAD [11] and used in more than 15 papers [12]. The traces contain around 4000 IP addresses, with highly bursty and intermittent behavior; in particular, over the 4 hours captured by the traces, most devices only appear for a few minutes, before disconnecting forever. Out of these 4000 IP's, we decided that the 10 highest-volume devices would run PriFi (the cost of using PriFi typically increase with the presence of high-volume traffic), and we manually assigned them to PriFi clients.

To also model the cost of anonymizing *someone else*, we added a dummy node to the simulation, i.e. a node that did not appear in the original traces, but that will participate in the simulation. This node will experience the highest cost of anonymity, as he does not uses the anonymous network himself, but sends chaff traffic to anonymize others.

PriFi policies. We fix the round-sharing policy to be 1 round per client every N rounds, where N is the number of clients. We (arbitrarily) fix that the *downstream bandwidth* will be 10 times higher than the *upstream bandwidth*, to better match the downstream/upstream ratio usually found in web applications[8].

Finally, we implement a naïve *round-reservation* mechanism: we reserve the first N bits of each PriFi payload, and allow clients to set a flag if they want to continue communicating. There is a minimum of 1 round per second; during this second, as soon as all reservations bits are 0, all devices sleep until the end of the second. Without this policy, long idle period in the real input traffic resulted in chaff traffic being sent as usual, with *no one* using the anonymous network. This is a simple change that reduce this waste, and more aggressive options can be experimented (e.g. exponential decrease of number of rounds per second).

[8] Here the number 10 is arbitrary, and models an order of magnitude above the upstream traffic.

Experiment. We vary the number of round/second between 0 and 200; we compute the size of the *cell* (the PriFi payload in one round) such that the bandwidth used does not exceed the network capacity. We do not fix the amount of bytes sent per second, but allow devices to reduce the bitrate when no one needs to communicate.

Experiment results. The figure 3 shows the results of the simulation. The blue line in Figure 3(a) depicts the number of messages (an important metric, since the radio interface of cellphones need to wake up to send any message), whereas the red line depicts the number of bytes sent, which is positively correlated with the transmission power used. In Figure 3(b), the added latency is shown for the same experiment.

We immediately see the trade-off between latency and number of messages; the first decreases when the latter increases. We see that the number of bytes increase up to a limit, which is linked to the base traffic and the number of users.

Furthermore, we see that in this scenario, for 85 rounds/sec, we add latency in the order of magnitude of 100ms, for an increase of 3 times the number of messages, and 3 times the number of bytes sent. This is *costly*, but tolerable: in addition, remember that the scenario is disadvantageous for PriFi (nodes are connected for a fraction of the total time in the real traces, but are considered connected for the whole duration in the simulation).

More detailed results can be found in Tables 1, 2 and 3. All tables show the difference between the real traces (before running PriFi) and the anonymized traffic (going through PriFi). Table 1 shows this difference for one of the 10 high-volume node selected manually in the traces; Table 2 show this increase for a *dummy* node, i.e. this is the biggest increase possible in that scenario; finally, Table 3 shows the difference from the point of view of the network, i.e. aggregated for all devices.

Table 1 Pre- and Post-PriFi comparison, for a random high-volume node

	Non-PriFi	PriFi	Increase
number of messages	13'568	202'204	14.91x
number of bytes sent	18 MB	106 MB	5.85x
median latency	0	339 ms	-
mean latency	0	476 ms	-

Table 2 Pre- and Post-PriFi comparison, for a dummy device

	Non-PriFi	PriFi	Increase
number of messages	0	202'204	∞
number of bytes sent	0	106 MB	∞

4.3. Discussions

We now have a much better idea on the cost of anonymization through PriFi. The current results of the simulation clearly show that running PriFi will incur important costs; however, to put these results in perspective, **(1)** we clearly did not simulate the best-case scenario, or any scenario where PriFi would perform naturally well, and **(2)** we simulated the current version of PriFi, which is still at the stage of prototype. It seems reasonable to obtain better performance by tuning and improving the prototype (e.g. better scheduling policy, allowing all devices to sleep).

In addition, we now have a way to tune PriFi (in terms of settings, scheduling policies, etc) *a priori*, as long as we have a set of traces that corresponds to a situation. PriFi-Sim allows us to evaluate the performance and cost of deploying PriFi in a given scenario, and this will be a powerful tool to model the different scenarios envisioned (IoT sensor network, corporate network, university network, etc.)

5. Engineering aspects

In the design of PriFi, we mentioned tailoring our solution to LANs and WLANs. This does not only come from the high bandwidth usually present in those networks (although it is of great importance due to the high bandwidth requirements of DC-nets); one other key property of WLANs (and certain LANs, depending on the topology) is the *inherent, low-cost broadcast of messages*.

In a typical WLAN, the messages from one point to another are effectively broadcasted already, only to be ignored by the clients not included in the communication. Using broadcast in this scenario is as simple and inexpensive as changing the destination IP address of the packet sent.

Broadcast & PriFi. In PriFi, broadcasting the messages from the relay to the client is an efficient way to save bandwidth and time. At the application level, downstream messages *must* be broadcasted to avoid equivocation attacks from the relay. Hence, at the network level, broadcasting is much cheaper, and takes less time, than unicasting sequentially the same message to each client.

The previous prototype of PriFi [3] used exclusively on point-to-point communication through TCP. To support broadcast, we needed to implement UDP support. Unfortunately, the code of the prototype was particularly unfit for UDP, as out-of-order messages were not handled at all, and imposing an order would require an expensive machinery.

Table 3 Pre- and Post-PriFi comparison, aggregated for all devices

	Non-PriFi	PriFi	Increase
number of messages	699'727	2'224'244	3.18x
number of bytes sent	512 MB	1480 MB	2.89x
median latency	0	334 ms	-
mean latency	0	820 ms	-

The main reason is that the prototype's code was inherently *sequential*. A much better approach for handling out-of-order message is to have an *event-driven* code.

5.1. Sequential vs Event-driven code

Sequential code is usually best for iterative programs, that do actions one-at-a-time, step-by-step. Those programs are easy to model using flowcharts or state machine diagrams, and much simpler to build, as the logic of the actions follows the order of the code. Unfortunately, all actions must proceed according to a pre-defined sequence. In our scenario, the PriFi prototype needs to handle messages, and reacts given the type and the content of these messages. As long as those messages are kept in sequence, and no message is lost, *sequential programming* is a perfectly valid coding paradigm. In addition, it keeps the structure of the code simple, as the actions that follow each others in the code happens consecutively.

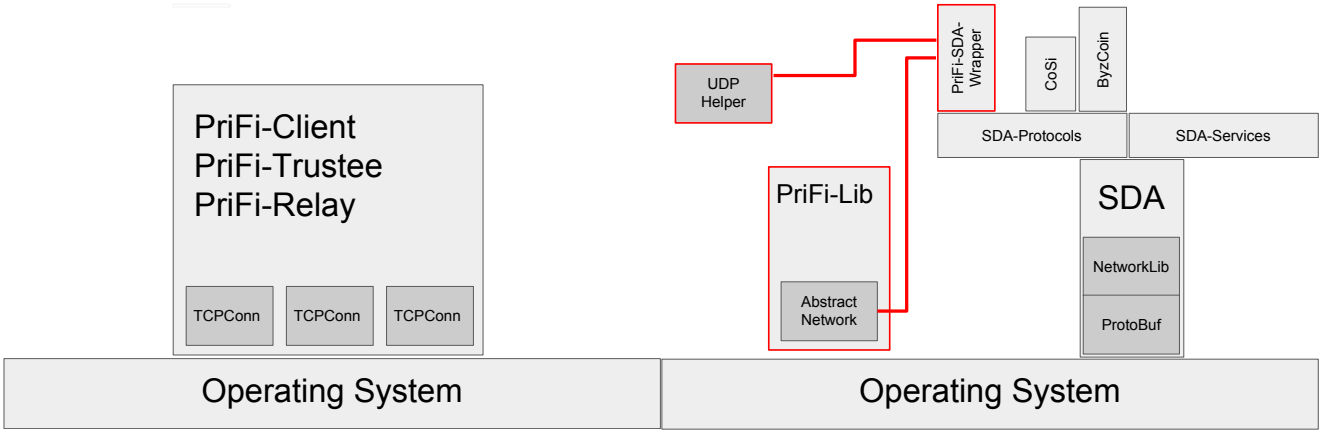
As mentioned above, our program needs to handle not only in-sequence messages, but also the out-of-order messages inherent to UDP. In that situation, *event-driven programming* is better suited, since by definition event can occur in any order. This paradigm forces the developer to keep manually the state of the program, and the events access and alter this state in any order. The programmer can declare a sequence of event invalid, but by definition no order is assumed, fitting our scenario where we receive UDP messages.

5.2. Rewrite of PriFi code

As explained, enabling broadcast in the PriFi prototype required a major rewrite of the code. We seized the opportunity to introduce additional layers of abstraction to simplify the code, and increase the maintainability.

SDA Framework. We relied on the Secure Distributed Algorithm [16] framework, developed by the DeDiS laboratory [15]. This framework handles the deployment of arbitrary *protocols* and *services* on a set of nodes. It handles the configuration of the network, organizes the nodes to increase communication performance, and performs all the necessary identities exchanges. Furthermore, it handles the coding and decoding of arbitrary messages, which was a significant burden in the original PriFi prototype. Finally, the framework also handle different kind of deployments: either on the local machine for testing purposes and quick debugging, or on the DeterLab [17] infrastructure. This allows to benchmark a protocol in realistic conditions with minimal effort.

PriFi-Lib. The SDA framework gives us an abstraction of the network topology. To further simplify the code of Prifi, and isolate the logic of the protocol, we created an additional intermediate step abstracting fully the network and the SDA from PriFi. The result is a



(a) Previous prototype

(b) New prototype

Figure 4: The architectures of the previous and new code of PriFi's prototype.

high-level library *PriFi-lib*, which has no knowledge of the network and how to communicate between entities, but concentrates on the higher-level logic. This approach brings several advantages: the core logic represents a much smaller code (3200 lines of code) than the previous prototype (5600 lines of code), is easier to maintain, and is not dependent on the underlying framework. This last point is of capital importance for enabling easy collaboration: PriFi-lib being independent of SDA, a new contributor can work with the core of PriFi without learning all the complex SDA specificities.

Figure 4 depicts the differences between the old and the new code. Figure 4(a) shows the previous PriFi prototype, which did not use any framework. Figure 4(b) shows the new standalone *PriFi-lib*, which is network-agnostic; the connection to SDA stack, called *PriFi-SDA-Wrapper*; the UDP component, *UDP-Helper* (introduced in Subsection 5.3); and finally the SDA stack, made of various sub-components such as NetworkLib, used to communicate between hosts, and ProtoBuf, used to encode messages.

New challenges. Interesting challenges arose when testing the new code. Since decoding the messages is now done by the SDA framework, we cannot use anymore the rate-limiting feature of TCP: buffers are emptied as soon as possible by the SDA framework, preventing the remote host from knowing when to stop.

To prevent the relay from being flooded by messages from the trustees, we added application-level awareness of the relay's ciphertext buffers. The relay stores up to W ciphertexts per trustee, and sends *control messages* to each trustee indicating the status of their buffer. When a buffer reaches a critical full rate W_{Max} (defined as 90% of W) the relay indicates to the trustee to stop sending. The relay tells the trustee to start sending at W_{Min}, which we define as follow:

$$W_{\text{Min}} = (W_{\text{Max}} - 1) - R \cdot (W_{\text{Max}} - 1)$$

For a ratio $R = 0$, as soon as the buffer leaves the critical state (fill rate above W_{Max}), the relay asks the trustee to resume sending ciphertexts. For R increasing, the relay waits more, emptying its buffers. On one hand, there is a notion of *risk*: if W_{Min} is too low, and the delivery of the control message takes more time than expected, the relay could reach a situation where **(1)** all clients are ready to finish a round, and **(2)** at least one of the trustee is not, blocking the round. This would have a catastrophic effect on the perceived latency for the clients, and the algorithm should avoid this at all cost. On the other hand, a higher value of W_{Min} means sending more control message per round and per trustee, as the effective window is smaller and leads to more oscillations between W_{Min} and W_{Max}.

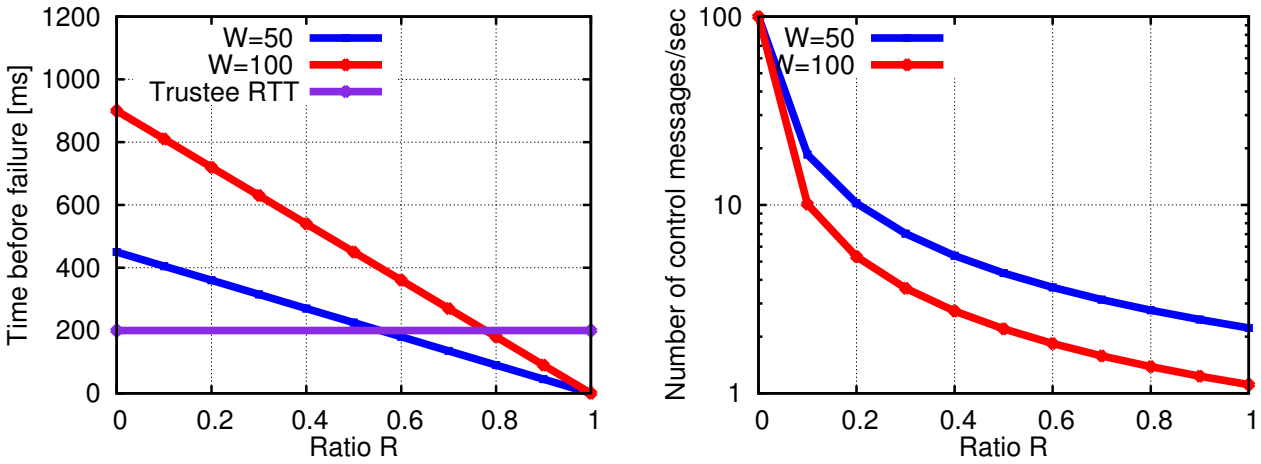
We analytically explore the behavior of our algorithm given the ratio R. We assume $M = 5$ trustees, $W_{\text{Max}} = 0.9W$, and 100 rounds per second. Figure 5 shows the results of this analysis. Figure 5(a) show the time margin the relay has before missing a cipher, starting when the algorithm sends a "resume sending" message. The purple line "Trustee RTT" is the time to contact a trustee, plus the time to get a ciphertext for answer, neglecting the computation time. Any value below this line means that the relay *will* wait on the trustee, a highly undesirable situation. Figure 5(b) shows the number of control messages sent to maintain the buffer between the appropriate bounds. While those control messages are short (TCP message with 4 bytes of payload), this value should be minimized.

The buffer size W translates directly into a storage requirement for the relay. Assuming a full buffer, a ciphertext size of 6 KB, and 5 trustees, the relay needs to store up to 1.46 MB and 2.92 MB for $W = 50$ and $W = 100$, respectively. In modern routers, this requirement is totally acceptable; furthermore, we see that by increasing W, we get both a *lesser risk* and *fewer control messages* for a given value of R. Hence, W should be maximized with respect to the memory (or storage) available.

5.3. UDP

The underlying motivation for writing a new code was the ability to broadcast messages. We seized the opportunity to improve the architecture of the code, but we have not discussed yet about the integration of UDP for broadcasting messages.

UDP through SDA. The most straightforward way to add UDP connection is to change the SDA itself, as it is responsible for all the networking in our architecture. This would allow minimal changes to PriFi-lib and PriFi-SDA-Wrapper, with the added benefit of improving the SDA (on which other protocol relies). Unfor-



(a) Duration before the relay misses a ciphertext.

(b) Number of control messages sent given R.

Figure 5: The effect of buffer size W and restart ratio R on *time before failure*, which happens when the relay is missing a ciphertexts he needs to complete a round, and on the *number of control messages* sent per trustee per second.

tunately, the complexity of SDA makes this change prohibitively costly. Hence, we explored the possibility to add a UDP "helper" to PriFi-SDA-Wrapper, bypassing the SDA.

The following trade-offs were identified : On one side, this is suboptimal, as **(1)** some of the messages go through the SDA framework, some through our UDP helper, which is conceptually not the simplest approach, and **(2)** it requires to re-implement functionalities already present in SDA, such as message encoding and decoding. On the other side, it will have advantages in terms of performance, since we can tune this helper to our specific problem (in contrast to the general-purpose SDA framework), and in particular this helper would be used for the biggest and most frequent messages, for which the SDA would probably not perform extremely well. Hence, we decided to code UDP in a external helper, as shown in Figure 4(b).

Local emulation of UDP. Unfortunately, network cards usually ignore broadcast messages when the message originates from the same computer. In practice, it means that broadcast messages sent by a computed C are not received (by any interface) on the same computer C, and there is no easy fix[9]. Hence, our local scenario where multiple threads run on the same computer and try to communicate using broadcast does not work.

This is problematic, as the whole point of using abstraction layers (SDA, and PriFi-SDA-Wrapper) was to make the code work with *any* network (on localhost as well as on real networks).

To bridge this gap, and allow our program to work seamlessly either on localhost or in a distributed environment, we coded an emulation of the broadcast using Go channels. It allows 1:N communication between treads, and is conceptually similar to UDP; it is still unrealistic, as the information stay in the computer's memory, and there is no re-ordering of messages (message loss on the contrary is artificially recreated).

[9]It is theoretically possible by setting a LOOPBACK flag on the interface, but this makes the code interface-dependent, and no easy way exists in Go.

6. Conclusion and Future work

In this work, we explored an important novel aspect: PriFi on resource-limited, mobile nodes. We built a general-purpose simulator, PriFi-Sim, that indicated the (partial) practicality of PriFi in the mobile world. We argued that at this stage of prototyping, the results were satisfactory enough, and hinted possible optimizations to further improve the practicality of the approach. Finally, we discussed how PriFi-Sim allows quicker testing of PriFi policies.

We also presented major changes in the prototype, and justified the engineering considerations behind them. We discussed how those changes were needed to improve the scalability and efficiency of PriFi. We highlighted a novel problem that arose with the changes, a presented a possible solution.

Future work. To continue on the engineering topic, the new prototype is not ready yet to handle real traffic : recoding a VPN (or SOCKS5 Proxy) should be one of the next steps. Then, performance of the new prototype needs to be assessed, as we do not have metrics on the overhead of the SDA framework. Following the same idea, no actual benchmark of the performance gain of using UDP has been made; it would be interesting to model the behavior of UDP in PriFi, e.g. with respect to the error rate on the links, or with respect to the PriFi bitrate.

On the protocol level, the most urging problems are perhaps **(1)** the absence of encryption (clients see each other's traffic, and clients have no way of recognizing their own traffic), **(2)** the possibility for the relay to mount *equivocation attack* (since clients do not yet check and agree on the history of communication), and **(3)** the possibility to for client to corrupt a round without being detected (since no mechanism punishes a client that sends a malformed message).

Acknowledgments. We wish to thank Mohamad El Hajj for his help with the implementation and with Section 5.

References

1. Matthew M. Aid. Nsa surveillance programs cover 75% of internet traffic transiting u.s., 2013. URL: http://www.matthewaid.com/post/58904880659/nsa-surveillance-programs-cover-75-of-internet.

2. Adam Back, Ulf Möller, and Anton Stiglic. *Traffic Analysis Attacks and Trade-Offs in Anonymity Providing Systems*, pages 245–257. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. URL: http://dx.doi.org/10.1007/3-540-45496-9_18, doi:10.1007/3-540-45496-9_18.

3. Ludovic Barman. Prifi: a traffic-analysis resistant, low-latency anonymous wlan. *EPFL Technical Report*, 2015.

4. Ludovic Barman. Prifi-sim, a network simulator for prifi, 2016. URL: https://github.com/lbarman/prifisim_dev.

5. Laura Poitras Barton Gellman. U.s., british intelligence mining data from nine u.s. internet companies in broad secret program, 2013. URL: https://www.washingtonpost.com/investigations/us-intelligence-mining-data-from-nine-us-internet-companies-in-broad-secret-program/2013/06/06/3a0c0da8-cebf-11e2-8845-d970ccb04497_story.html.

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

7. David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

8. Chen Chen, Daniele E Asoni, David Barrera, George Danezis, and Adrain 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.

9. Henry Corrigan-Gibbs and Bryan Ford. Dissent: accountable anonymous group messaging. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 340–350. ACM, 2010.

10. CRAWDAD. Dataset of wireless lan traffic around portland, oregon using a commercial sniffer vwave, 2009. URL: http://crawdad.org/pdx/vwave/20090704/.

11. CRAWDAD. A community resource for archiving wireless data at dartmouth, 2016. URL: http://crawdad.org/.

12. CRAWDAD. Papers quoting "dataset of wireless lan traffic around portland, oregon using a commercial sniffer vwave", 2016. URL: http://www.citeulike.org/group/5303/tag/pdx_vwave.

13. George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type iii anonymous remailer protocol. In *Security and Privacy, 2003. Proceedings. 2003 Symposium on*, pages 2–15. IEEE, 2003.

14. Darlene Storm Darlene Storm. 75surveillance, 2014. URL: http://www.computerworld.com/article/2870773/75-of-writers-in-free-countries-self-censor-due-to-fears-of-mass-surveillance.html.

15. DeDiS. Dedis laboratory, epfl, 2015. URL: http://wiki.epfl.ch/dedis.

16. EPFL DeDiS Laboratory. Secure distributed algorithm plateform, 2015. URL: https://github.com/dedis/cothority/tree/master/sda.

17. DeterLab. Deterlab: Cyber-defense technology experimental research laboratory, 2016. URL: https://www.isi.deterlab.net.

18. Michael J Freedman and Robert 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.

19. Claire Davenport Georgina Prodhan. U.s. surveillance revelations deepen european fears of web giants, 2013. URL: http://www.reuters.com/article/europe-surveillance-prism-idUSL5N0EJ3G520130607.

20. Sharad Goel, Mark Robson, Milo Polte, and Emin Sirer. Herbivore: A scalable and efficient protocol for anonymous communication. Technical report, Cornell University, 2003.

21. Glenn Greenwald. Nsa collecting phone records of millions of verizon customers daily, 2013. URL: http://www.theguardian.com/world/2013/jun/06/nsa-phone-records-verizon-court-order.

22. Erik Hjelmvik. China's man-on-the-side attack on github, 2015. URL: http://www.netresec.com/?page=Blog&month=2015-03&post=China%27s-Man-on-the-Side-Attack-on-GitHub.

23. Hsu-Chun Hsiao, TH-J Kim, Adrian Perrig, Akimasa Yamada, Samuel C Nelson, Marco Gruteser, and Wei Meng. Lap: Lightweight anonymity and privacy. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 506–520. IEEE, 2012.

24. Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users get routed: Traffic correlation on tor by realistic adversaries. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 337–348. ACM, 2013.

25. Benjamin Edelman Jonathan Zittrain. Empirical analysis of internet filtering in china, 2005. URL: http://cyber.law.harvard.edu/filtering/china/appendix-tech.html.

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

27. Stevens Le Blond, David Choffnes, Wenxuan Zhou, Peter Druschel, Hitesh Ballani, and Paul Francis. Towards efficient traffic-analysis resistant anonymity networks. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 303–314. ACM, 2013.

28. C Andrew Neff. Verifiable mixing (shuffling) of elgamal pairs. *VHTi Technical Document, http://www. votehere. net/vhti/documentation/egshuf. pdf, VoteHere, Inc*, 2003.

29. The Tor Project. Traffic correlation using netflows, 2014. URL: https://blog.torproject.org/blog/traffic-correlation-using-netflows.

30. The Tor Project. Tor project: Anonymity online, 2016. URL: https://www.torproject.org/.

31. Xiao Qiang. How china's internet police control speech on the internet, 2008. URL: http://www.rfa.org/english/commentaries/china_internet-11242008134108.html.

32. Michael K Reiter and Aviel D Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security (TISSEC)*, 1(1):66–92, 1998.

33. Paul Syverson Roger Dingledine, Nick Mathewson. Tor: The second-generation onion router, 2004. URL: https://svn.torproject.org/svn/projects/design-paper/tor-design.html.

34. Jeff Sharkey. Coding for life - battery life, that is (google io 9), 2009. URL: https://dl.google.com/io/2009/pres/W_0300_CodingforLife-BatteryLifeThatIs.pdf.

35. Li Sun, Ramanujan K Sheshadri, Wei Zheng, and Dimitrios Koutsonikolas. Modeling wifi active power/energy consumption in smartphones. In *Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on*, pages 41–51. IEEE, 2014.

36. P. Palfrader L. Sassaman U. Moeller, L. Cottrell. Mixmaster protocol version 2, 2004. URL: https://tools.ietf.org/html/draft-sassaman-mixmaster-03.

37. Wired. The untold story of silk road, 2015. URL: http://www.wired.com/2015/04/silk-road-1/.

38. David I Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Scalable anonymous group communication in the anytrust model. Technical report, DTIC Document, 2012.

39. David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *OSDI*, pages 179–182, 2012.

40. Bassam Zantout and Ramzi Haraty. I2p data communication system. In *Proceedings of the tenth international conference on networks*, pages 401–409, 2011.