Distributed Security Infrastructures: Project Report Winter Term 2022/23

Theresa Tratzmüller
TU Berlin
Berlin, Germany
t.tratzmueller@tu-berlin.de

ABSTRACT

In this project, we develop a mechanism for introducing micro-delays into *Tor* for the purpose of traffic pattern obfuscation. This mechanism in turn is intended as a building block for future work towards and answer to whether stronger anonymity guarantees in the Tor network can be achieved by means of a strategy combining cover traffic with micro-delays.

1 INTRODUCTION

The *Tor* network is an overlay network providing low-latency anonymous communication for TCP-based applications [5]. With Tor, the link between communicating parties is obscured by routing traffic via so-called circuits composed of three Tor relays. Crucially, of all the nodes along a circuit traversing it a relay will only know its predecessor and successor. By design, Tor only provides protection against adversaries with a partial view of the network, e.g., a national internet service provider [5]. A *global passive adversary (GPA)* that is able to observe the entire Tor network can match traffic patterns to determine who is talking to whom [6, 11].

A *Mix network*, e.g., *Mixminion* [2] or *Vuvuzela* [15], is another type of anonymous communication system whose anonymity guarantees rely on routing traffic through one or multiple intermediate nodes. Unlike Tor, these networks are designed to defend against GPAs. To this end, they employ strategies such as generating cover traffic and forwarding traffic in discrete rounds in order to destroy timing patterns. They are thus normally used with message-oriented, delay-tolerant applications such as e-mail and regarded as rather unsuitable for use with applications requiring real-time interactivity such as web browsing or instant messaging.

With *Loopix*, Piotrowska et al. [12] present an anonymity system that is able to defy global passive adversaries as well as active attacks while still supporting low-latency traffic. This is achieved by means of the *Poisson Mixing Strategy*, which is composed of two mechanisms: First, traffic entering the network is shaped to follow a Poisson process. Different types of cover traffic are generated in order to achieve this. Second, independent micro-delays sampled from an exponential distribution are applied to the forwarding of messages. As a result of the overall strategy, a message emitted by an intermediate node is indistinguishable within a set of messages previously received by that node. *Katzenpost* [1] is a free software implementation of the Loopix design for use with message-oriented applications.

Since the goal of all anonymity systems is to hide users among users, attracting a critical mass of users is a key concern [4]. Tor has by far been the most successful in this regard. Integrating an optional sub-protocol with stronger anonymity guarantees into Tor would allow for exploitation of the network's large user base while supporting usage scenarios for whom regular Tor is not sufficiently anonymous. Our present work is thus part of a larger effort to determine how the Loopix protocol might be integrated into Tor.

Supporting the Loopix protocol in Tor requires the ability to generate cover traffic as well as to artificially insert forwarding delays. While Tor includes a framework for cover traffic generation since release 0.4.0.5 [10], the introduction of delays is not supported thus far. In this report, we describe the implementation of a mechanism for inserting forwarding delays (§2), evaluate this mechanism (§3), and provide next steps towards the overarching goal of integrating Loopix into Tor (§4).

2 IMPLEMENTATION

Our implementation is an adapted version of the Tor source code [13].

2.1 Core mechanism

So-called *cells* of fixed 514 byte length constitute the atomic units of communication in Tor. As our goal is to obfuscate patterns of actual application traffic, our delay mechanism is limited to cells of the *relay* type. This cell type is the one used for exchanging data once communication has successfully been established.

In Tor, a *circuit* represents a path over the onion routing network. Our mechanism for the introduction of delays is located on this layer of abstraction. At each Tor node, cells to be relayed on a circuit are appended to a cell queue. There is one queue for each direction of traffic (*exitward*, i.e., client to destination, vs. *inward*, i.e., destination to client). From there, they are dequeued for writing to the network according to a scheduling policy. Our mechanism for delay introduction is currently limited to cells traveling in the exitward direction (§2.3). It relies on an additional cell queue, referred to in the following as the *delay queue*. Each cell to be relayed in the direction from client to destination is initially appended to this queue instead of the regular exitward queue. After the respective delay has elapsed, a callback function is activated and one cell is transferred from the delay queue to the regular exitward queue.

2.2 Sampling delays

Per default, forwarding delays in Loopix are sampled from an exponential distribution with scale parameter 0.001[1]. As the scale parameter of an exponential distribution corresponds to its mean and the sampled values are treated to be in units of seconds, the resulting mean forwarding delay per hop is 1 millisecond.

In Tor, this behavior can be realized using the natively supported *Weibull* distribution, of which the exponential distribution is a special case. Parameterizing it with $k = 1$ and $\lambda = 0.001$ allows for sampling from an exponential distribution with scale parameter 0.001.

Note, however, that the mean delay introduced in Loopix is not fixed, but a tweakable parameter that has to be considered in conjunction with the respective traffic rate in order to determine the level of privacy achieved by the overall mixing strategy [12]. In this work we decided to stick with the default value of 0.001.

2.3 Sending instructions

The Loopix system architecture employs the *Sphinx* packet design [3, 12], in which each encrypted message is composed of a hop-specific header and a payload. Decryption at a hop yields routing information (resulting from decryption of the header) as well as a new packet to forward to the next hop (resulting from decryption of the payload). The Loopix authors extended the Sphinx format by including additional per-hop routing instructions, most notably the delay to introduce when forwarding the respective message. The client is thus responsible for sampling delays and able to verify if intermediate hops apply them as instructed.

Tor cells are composed of a circuit ID, a command, and a payload. The circuit ID allows intermediate hops to look up the next hop to forward to, while the command (e.g., `CELL_RELAY`) indicates how the payload should be processed. In the case of relay cells, the payload consists of the relay header and relay body. Unlike Loopix, Tor uses the same relay header for all hops. The only component of the relay header that differs between hops is the `recognized` field, which enables hops to detect whether the cell is fully decrypted after tearing off their layer of encryption. If it is, its contents are processed at the hop. Otherwise, it is forwarded along the circuit.

[1]https://github.com/UCL-InfoSec/loopix/blob/master/loopix/config.json

Due to the absence of hop-specific relay headers in the Tor protocol, we decided to introduce a dedicated relay cell type for the purpose of the client providing the hops along the circuit with sampled delay values. The delay to introduce when forwarding a cell at a hop is thus not passed on with that cell itself, but rather taken from an array of delay values received from the client at an earlier point in time. Specifically, the client supplies each hop along the respective circuit with such a fixed-size array of sampled delays before sending the first relay cell. Subsequently, it will only send as many relay cells as there are values contained in these arrays before sending a fresh supply. Concretely, this design was realized by specifying a custom binary format for transferring the delay arrays across the network using the *Trunnel* framework [14]. We leverage Tor's *leaky-pipe* circuit topology for sending cells to intermediate hops on a circuit [5].

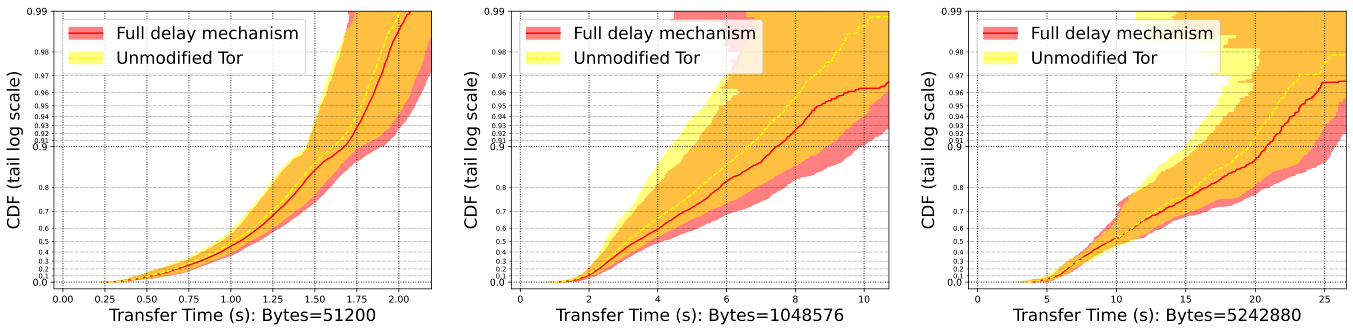
Yet, observe that this mechanism only covers the traffic direction from client to destination. The client is only able to ensure that fresh delays are always available for cells originating from its end of the circuit. This is simply due to the fact that the client can precede the forwarding of a relay cell with the sending of a delay array should the previous supply have been exhausted. For the opposite direction, i.e., traffic traveling from destination to client, the client is clearly not able to prevent relay cells from being sent once there are no unused delays left. Extending the described mechanism to the opposite direction would require the last onion router in the circuit to provide the intermediate hops with delays to apply for this direction. This is impossible as by design, only the client knows all the other hops in the circuit and shares a secret key with each of them. Thus, no other node along the circuit is able to make use of the leaky-pipe topology.

3 EVALUATION

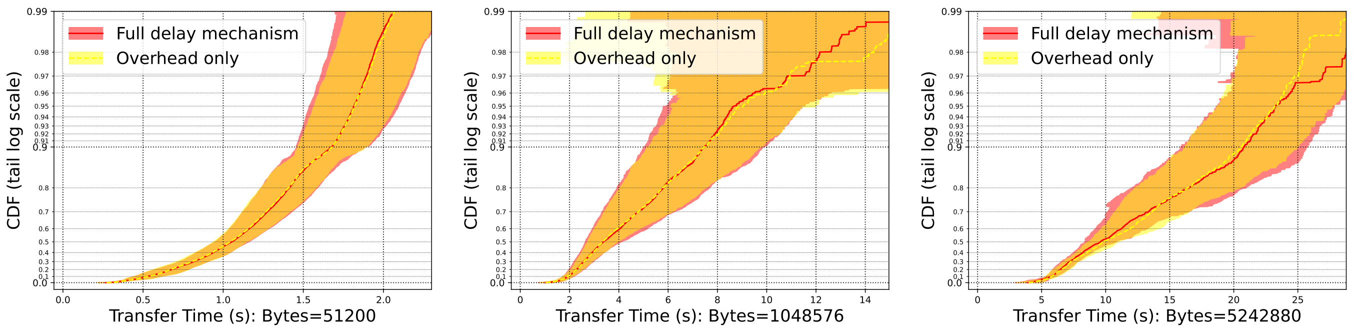
We verify the behavior of our implementation with *tornettools* (v1.1.0), a *Python*-library for generating realistic private Tor network models [9], as well as the *Shadow* discrete-event network simulator (v2.3.0) [7, 8]. We use *TGen* (v1.1.1) to simulate traffic flow in Shadow. Our evaluation is conducted on an `AWS EC2` instance of type `z1d.large` running Debian 11.

As stated previously, the present work merely constitutes a preliminary technical study in preparation of future research. Therefore, no analyses of privacy properties were performed. We simply verified that including the described mechanism in Tor would produce the intended delays in traffic from client to destination.

Three variants of the Tor codebase were contrasted in the evaluation:



(a) Transfer time as compared to the unmodified version of Tor.



(b) Transfer time as compared to a delay of 0 μs.

Figure 1: Transfer time from the perspective of file-servers, compared between our Tor modification and an unmodified version of Tor (Fig. 1a) and our Tor modification with a delay of 0ms (Fig. 1b).

- Tor including the entire mechanism for introducing delays supplied by clients as described above. The client supplies arrays of 32 sampled delays at a time.
- Tor including all the overhead introduced by the described mechanism, but with static zero delays.
- Unmodified Tor without delays.

Six Tor networks at the scale of 0.01% of the actual Tor network were generated using tornettools. Note that in the context of analyzing complex system properties, e.g., the level of privacy, a sample of six scaled-down networks would generally be considered too small [9]. Yet, for our modest goal of mere functionality verification it can be deemed sufficient. We specify none of the optional parameters available for network model generation. Each of the three Tor codebase variants listed above was used to carry out one tornettools simulation on each of those networks. Each simulation was performed in the default configuration (i.e., without supplying optional parameters) and using the same seed that had already been used to generate the respective network.

By default, a tornettools simulation consists in a number of clients downloading files in three different sizes (50 KiB, 1 MiB, and 5 MiB) from a set of file-servers via the Tor network for the duration of one simulated hour [9]. Such a scenario in which the majority of the data exchanged is sent from server to client can be regarded as representative of the behavior of applications typically used with Tor. Still, it is not very useful for analyzing the effect of our delay mechanism, which is currently limited to traffic traveling from client to server. Our simulations therefore use an adapted traffic flow scenario in which clients upload 50 KiB, 1 MiB, and 5 MiB files rather than downloading them.

tornettools collects several metrics during the course of an experiment. The one most relevant to our evaluation is the *time to last byte*, defined as the amount of time it takes to complete a download. Yet, tornettools expects files to be downloaded by clients from servers. The insights and plots automatically generated by the library are therefore not suitable for displaying our results. We thus employed TGen's *tgentools* utility to retrieve the *time to last byte* measurements from the perspective of the file servers and modified the tornettools code for the purposes of our analysis.

The results are displayed in Figure 1. Solid lines represent estimates of the true cumulative density functions (CDFs), while the shaded areas show the associated confidence intervals at $\alpha = 95\%$ confidence [9]. The full delay mechanism appears to indeed produce longer transfer times as the unmodified Tor codebase. Regarding the comparison between the variant including the full delay mechanism and its overhead-only counterpart, the results are inconclusive. It appears that for the tiny delays introduced in the experiment (mean 1ms), the mechanism's overhead has a more prominent effect on transfer times than the actual delay introduced.

4 DISCUSSION & OUTLOOK

While the entire mechanism produces larger transfer times as unmodified Tor, the pure overhead incurred by sending and applying delays seems to dominate the actual delays in explaining this effect. Given that the introduced delays were very small, it is desirable to repeat the experiment with larger delays. It might also be worthwhile to increase the size of the delay arrays so that a fresh supply has to be sent less frequently.

The mechanism presented is currently limited to traffic originating from a Tor client and traveling towards a destination (e.g., a Web server). In the case of the types of applications typically used with Tor, the majority of exchanged data is however transferred from the server to the client. The utility of the mechanism in its current form is therefore limited. The next step in this ongoing research is thus to extend it to cover the other direction of traffic as well. Due to its unique ability to exploit the leaky-pipe topology only the client is able to supply intermediate hops in a circuit with sampled delays. A promising strategy to explore hence consists in harnessing Tor's congestion control functionality to ensure that the destination can only send as many cells as there are unused delays before the client provides a fresh supply.

Further, it is questionable if introducing delays on a per-cell basis as is currently done is appropriate. The crux here lies in the fact that the Loopix protocol is designed for message-oriented applications and delays are thus introduced on a per-message basis. Applications commonly used with Tor by contrast are stream-based, and a cell constitutes a mere chunk of stream data with no inherent semantic meaning. Transferring Loopix' behavior to Tor requires determining how continuous streams of data can be conceptually divided into discrete messages. For example in the case of Web browsing, a set of cells carrying data of the size of a typical Website could be regarded as a message.

When communicating over Loopix, a path of relay nodes across which to traverse the network is chosen independently for each individual message [12]. By contrast, Tor multiplexes multiple TCP streams along each circuit and clients only rotate to a new circuit periodically [5]. Resolving this heterogeneity poses another challenge regarding the incorporation of Loopix into Tor.

Next, in order to realize Loopix' *Poisson mixing* strategy, traffic entering the network has to follow a Poisson process. This requires introducing different types of cover traffic. Leveraging Tor's cover traffic framework for this purpose accordingly is a central concern of future research.

5 CONCLUSION

We have described the motivation behind as well as implementation and evaluation of a mechanism for introducing forwarding delays into Tor. A number of technical and conceptual challenges remain to be overcome.

As the mechanism presently only covers one of two directions of traffic and its effect on transfer times seems to mostly be due to the induced overhead, further engineering work is required to improve it. Additional conceptual work is necessary to determine the unit of communication to apply delays to. Finally, the introduction of delays has to be combined with cover traffic to achieve the full Loopix design.

REFERENCES

[1] Yawning Angel, George Danezis, Claudia Diaz, Ania Piotrowska, and David Stainton. 2017. Katzenpost mix network specification. (2017).

[2] George Danezis, Roger Dingledine, and Nick Mathewson. 2003. Mixminion: Design of a type III anonymous remailer protocol. In *Proceedings of the 2003 Symposium on Security and Privacy, 2003.* IEEE, New York, NY, USA, 2–15.

[3] George Danezis and Ian Goldberg. 2009. Sphinx: A compact and provably secure mix format. In *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy.* IEEE, New York, NY, USA, 269–282.

[4] Roger Dingledine and Nick Mathewson. 2006. Anonymity loves company: Usability and the network effect.. In *Proceedings of the WEIS.*

[5] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. *Tor: The second-generation onion router.* Technical Report. Naval Research Lab Washington DC.

[6] Jamie Hayes, George Danezis, et al. 2016. k-fingerprinting: A Robust Scalable Website Fingerprinting Technique.. In *Proceedings of the USENIX security symposium.* 1187–1203.

[7] Rob Jansen and Nicholas J Hopper. 2011. Shadow: Running Tor in a box for accurate and efficient experimentation. (2011).

[8] Rob Jansen, Jim Newsome, and Ryan Wails. 2022. Co-opting Linux Processes for High-Performance Network Simulation. In *Proceedings of the 2022 USENIX Annual Technical Conference (USENIX ATC 22).* 327–350.

[9] Rob Jansen, Justin Tracey, and Ian Goldberg. 2021. Once is never enough: Foundations for sound statistical inference in Tor network experimentation. In *Proceedings of the USENIX Security Symposium.*

[10] Nick Mathewson. 2019. *New Release: Tor 0.4.0.5.* Retrieved March 10, 2023 from https://blog.torproject.org/new-release-tor-0405

[11] Steven J Murdoch and George Danezis. 2005. Low-cost traffic analysis of Tor. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy (S&P'05).* IEEE, New York, NY, USA, 183–195.

[12] Ania M Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. 2017. The loopix anonymity system. In *Proceedings of the 26th USENIX Security Symposium (USENIX Security 17).* 1199–1216.

[13] The Tor Project 2022. *Tor's source code (commit* `5f548f05d2c5e18833358ae05b78ad3c3c673636`*).* The Tor Project. Retrieved March 10, 2023 from https://gitweb.torproject.org/tor.git/tree/?id=5f548f05d2c5e18833358ae05b78ad3c3c673636

[14] The Tor Project 2023. *Trunnel: a code generator for binary encoders/decoders.* The Tor Project. Retrieved March 10, 2023 from https://gitweb.torproject.org/trunnel.git

[15] Jelle Van Den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. 2015. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*. 137–152.