Tor-NetShaper Project Design

April 30, 2025

Section 1. Current Pluggable Transports Overview

. There are different pluggable transports that can be used to integrate NetShaper into Tor. The two more viable options at the moment are Obvsproxy and Lyrebird/Obfs4. The first has the advantage of being written in Python, and has already been extensively used to create modded PTs in a academic articles. The main problem is that it seems to be deprecated, the last update was rolled out 10 years ago, and it's written in Python 2.7. I tested it for a bit, and after some precarious adjustments I've been able to create a local server with it on my machine, but it was very unstable.
Obfs4 is the current state-of-the-art of Tor PTs, the main problem is that it is written in Go, a language that I don't know at all, and that from what I can see it's not been used before in the NetShaper project. Obfs4 has been recently moved into Lyrebird, a framework that implements several pluggable transport protocols, including obfs4, meek, Snowflake and WebTunnel. From now on I'll assume that the implementation of NetShaper will happen in Lyrebird, since it's the most modern and supported option. The structure of Obvsproxy is not particularly different, so it should be straightforward to eventually use that option.
There is another interesting library that could be useful. The goptlib is a basic example of a Tor PT created by the Tor Project to help developers create a new PTs. □

Section 2. NetShaper PT Design Model

. The implementation of NS discussed here has been designed to satisfy the Pluggable Transport Specification.
Since Lyrebird has been created to manage different PTs, implementing a new one in it should be fairly easy. The transports/ folder contains all the current implementations, and to add a new one is necessary to create a new folder containing the related code, and then add the configuration to the existing code. In this regard the commit to add SnowFlake could be taken as an practical example.
Let's now talk about the proper NetShaper implementation. Taking as an example the simple structure present in goptlib, we can model a PT as two main components, PT-client and PT-server. The process of connecting to the Tor Network via a PT is structured in the following way. The PT-client creates a SOCKS proxy for the Tor Browser to connect and establishes a TCP connection to the configured bridge address (the PT-server), then applies the obfuscation technique (in our case DP) to the traffic. The PT-server (placed outside the censored network) receives the obfuscated traffic, de-obfuscates and forwards it to a "classic"

Tor bridge. To receive the response, the process is carried out in reverse order.
To implement NetShaper as a PT, there are a couple of points that needs to be adjusted.
1) Tor doesn't support QUIC traffic, but it should not be a problem, since the traffic would work like this:
Tor-Client —TCP— PT-Client (obfuscation) —QUIC— PT-Server (deobuscation) —TCP— Tor-Bridge
NS is implemented as an additional connection between the client and the bridge, so the Tor circuit remains unaffected by it.
2) The NS PT should have a structure analog to the one following to mimic the implementation of others PTs:

```
netshaper/
|-- client/
|   |-- main              # Entry point for PT-client
|   |-- socks_handler     # SOCKS5 proxy and connections from Tor
|   |-- quic_sender       # Handles shaping + sending shaped data via QUIC
|-- server/
|   |-- main              # Entry point for PT-server
|   |-- quic_listener     # Listens for QUIC streams from clients
|   |-- tcp_forwarder     # Forwards traffic to the Tor bridge
|-- common/
|   |-- shaping           # PT-client side logic
|   |-- deshaping         # PT-server side logic
|   |-- dp_noise          # Gaussian/DP noise sampling utilities
|   |-- framing           # Packet/burst framing logic
|   |-- config            # Env parsing
|   |-- logging
|-- go.mod
```

The NetShaper folder could be inserted directly into Lyrebird (example above), or created in a different repository, that will then be called by small wrapper from Lyrebird (example below), like for example the Snowflake integration.

```
netshaper/
|-- netshaper
```

3) The NetShaper variables can be tuned accordingly with the Tor PT Specification using the env variable TOR-PT-SERVER-TRANSPORT-OPTIONS. It's a list of parameters, for reference in the torcc file it could be used like this:

```
ServerTransportOptions netshaper epsilon_T=1, delta_T=1, delta_W=1, T=1
```

□

Section P. otentially useful articles

. [Pan+16] [Uma+23] [SZ17] [Han+] □

References

[Pan+16] Ioana-Cristina Panait et al. "TOR - Didactic Pluggable Transport". In: *Innovative Security Solutions for Information Technology and Communications*. Ed. by Ion Bica and Reza Reyhanitabar. Cham: Springer International Publishing, 2016, pp. 225–239. ISBN: 978-3-319-47238-6.

[SZ17] Khalid Shahbar and A. Nur Zincir-Heywood. "An analysis of tor pluggable transports under adversarial conditions". In: *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2017, pp. 1–7. DOI: 10.1109/SSCI.2017.8280829.

[Uma+23] Zeya Umayya et al. "PTPerf: On the Performance Evaluation of Tor Pluggable Transports". In: *Proceedings of the 2023 ACM on Internet Measurement Conference*. IMC '23. Montreal QC, Canada: Association for Computing Machinery, 2023, pp. 501–525. ISBN: 9798400703829. DOI: 10.1145/3618257.3624817. URL: https://doi.org/10.1145/3618257.3624817.

[Han+] Hans Hanley et al. "DPSelect: A Differential Privacy Based Guard Relay Selection Algorithm for Tor". In: *Proceedings on Privacy Enhancing Technologies* 2019.2 (). DOI: 10.2478/popets-2019-0025. URL: https://par.nsf.gov/biblio/10107792.