QCSD: A QUIC Client-Side Website-Fingerprinting Defence Framework

Jean-Pierre Smith and Luca Dolfi, *ETH Zurich;*
Prateek Mittal, *Princeton University;* Adrian Perrig, *ETH Zurich*

https://www.usenix.org/conference/usenixsecurity22/presentation/smith

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

A Artifact Appendix

A.1 Abstract

The artefact consists of scripts for collecting several datasets of live-defended VPN network traces using the QCSD framework, simulating defended network traces, and performing machine-learning evaluations, in addition to the source code of the QCSD client library and test clients written in Rust and the datasets collected during the evaluation. The artefact requires at least 2 CPU cores and 4 GB of memory, however additional cores help greatly to reduce run times, as does access to GPUs. It requires python3.8, rust, and docker and was tested on Ubuntu 20.04. The artefact generates the plots present in the paper and allows running the machine-learning evaluations on the datasets from the paper to compare the resulting plots to those in the paper.

A.2 Artifact check-list (meta-information)

- **Compilation: rustc >= 1.51, publicly available**
- **Data set: included**
- **Run-time environment: root access, Ubuntu 20.04, docker, rust, python3.8, git/git-lfs**
- **Hardware: Wireguard VPN servers, GPU**
- **Run-time state: impacted by network throughput**
- **Execution: test → under 24 hours; full → data collection > 4 days with 3 VPN gateways and a 32 core server, machine-learning evaluations several days on an RTX 3060**
- **Security, privacy, and ethical concerns: network scanning, web-page crawling**
- **Metrics: recall, *r*-precision, Pearson's correlation, LCSS**
- **Output: plots, tables, see paper for expected results**
- **Experiments: automated setup, snakemake workflow**
- **How much disk space required (approximately)?: 60 GiB**
- **How much time is needed to prepare workflow (approximately)?: 1 hour**
- **How much time is needed to complete experiments (approximately)?: test → under 24 hours, full → upwards of 6 days**
- **Publicly available (explicitly provide evolving version reference)?: yes, `https://github.com/jpcsmith/qcsd-experiments/tree/v1.0.1`**
- **Code licenses (if publicly available)?: MIT, Apache-2.0**
- **Data licenses (if publicly available)?: Creative Commons Attribution 4.0 International**
- **Workflow frameworks used?: Snakemake**
- **Archived (explicitly provide DOI or stable reference)?: `10.3929/ethz-b-000565356`**

A.3 Description

A.3.1 How to access

The repository containing the scripts can be by cloned from the GitHub repository `https://github.com/jpcsmith/qcsd-experiments.git`, with tag v1.0.1 corresponding to the version of this appendix. The dataset collected during the paper, along with archived versions of the associated repositories and a virtual machine with the software dependencies installed, can be found under the DOI `10.3929/ethz-b-000565356`.

A.3.2 Hardware dependencies

Below we describe the hardware dependencies based on the various phases in the workflow.

Dataset collection The provided test configuration runs on a server with 8 CPU cores and 8 GB of memory, and with a single Wireguard VPN gateway running on the same host. The full collection utilised 3 VPN gateways (2 CPU cores and 2 GB memory is more than sufficient for each) and 12 VPN clients per gateway (36 in total) running on a server with 32 cores and 188 GB of memory. Each VPN client is is restricted to at most 2 CPU cores. Reduce the number of clients per gateway to use less cores at the cost of longer dataset collection times.

Machine learning (ML) evaluations The ML evaluations associated with the test configuration can be run on 8 cores or less. For the full configuration, at least 1 GPU is recommended such as an RTX 3060 or better.

A.3.3 Software dependencies

The following software dependencies are assumed to be already installed, and are installed in the provided VM:

- *Ubuntu 20.04 and bash*: All code was tested on a fresh installation of Ubuntu 20.04.
- *git, git-lfs*: Used to clone the code repository and install python packages.
- *Python 3.8 with virtual envs*: Used to create a Python 3.8 virtual environment to run the evaluation and collection scripts.
- *docker >= 20.10*: Used to isolate simultaneous runs of browsers and collection scripts, as well as to enable multiple Wireguard clients on a single host. The user must be able to manage containers without using sudo.
- *tcpdump >= 4.9.3*: Used to capture traffic traces. Must be configured to allow the non-root user to capture.
- *rust (rustc, cargo) == 1.51*: Used to compile the QCSD library and test client library written in Rust.
- *Others*: Additionally, the following packages are required to build the QCSD library and test client, and can be installed with the ubuntu package manager, apt: `build-essential mercurial gyp ninja-build libz-dev clang tshark texlive-xetex`

Other software dependencies, such as ansible and Wireguard, are installed automatically.

A.3.4 Data sets

The dataset from the collection performed in the paper has the DOI `10.3929/ethz-b-000565356` and can be downloaded and used as a starting point for running the evaluations. To do so, replace the `results/` directory in the cloned repository with the results directory found in the gzipped tar archive.

A.3.5 Security, privacy, and ethical concerns

The evaluation downloads thousands of web page HTMLs and associated resources. The scripts in the workflow avoid overloading servers by scheduling requests such that sequential requests to a domain are either delayed or interleaved with requests to different domains. Additionally, be aware of any regulations of your network provider regarding performing automated web browsing.

A.4 Installation

If using the provided VM image, change to the home directory of the vagrant user `/home/vagrant/`, otherwise change to the directory in which you would like to install the artefact.

1. Clone the repository `https://github.com/jpcsmith/qcsd-experiments.git` using `git clone`.
2. Change to the code directory and pull the additional resources `cd qcsd-experiments && git lfs pull`.
3. If you want to use a specific version of the repository, change to it now (e.g., `git checkout v1.0.1`).
4. Create a Python 3.8 virtual environment and activate it `python3.8 -m venv env && source env/bin/activate`
5. Upgrade the python package manager (`python -m pip install -U pip wheel`) and install required python packages `python -m pip install --no-cache-dir -r requirements.txt`.

Decide whether you want to run the experiments locally or distributed across multiple machines. The file `ansible/distributed` contains an example of the configuration required for running with remote VPN gateways and clients. The file `ansible/local` contains the configuration for running the experiments locally, and is used as an example for the following steps.

6. Set the `gateway_ip` variable in `ansible/local` to the non-loopback IP address of the host, for example, the LAN IP address.
7. Change the `exp_path` variable to a path on the (local) filesystem. It can be the same path to which the repository was cloned.
8. Run the command `ansible-playbook -i ansible/local ansible/setup.yml` to setup the docker image for creating the web-page graphs with Chromium; create, start, and test docker images for the Wireguard gateways and clients; and download and build the QCSD library and test clients.

The QCSD source code is cloned on the remote host into the third-party/ directory of the folder identified by the `exp_path` variable in the hosts file (`ansible/local` or `ansible/distributed`).

A.5 Experiment workflow

Before running the workflow, it is necessary to ensure that the appropriate environment variables are set. This can be done with `source env/bin/active` to activate the python environment created during installation, and `source env_vars` to set environment variables for the project.

The results and plots in the paper were produced using snakemake. Like GNU make, snakemake will run all dependent rules necessary to build the final target. The general syntax is

```
snakemake -j --configfile=<filename> <rulename>
```

where `<filename>` can be `config/test.yaml` or `config/final.yaml` and `<rulename>` is the name of one of the snakemake rules found in `workflow/rules/*.smk` files or the target filename. Table 1 lists the figures in the table and the rules to produce them, whereas the following section describes the results in the paper and the rule used to produce them. The listed output files can be found in the results directory.

Generally, the various result workflows can be divided into the phases: scan, collect, evaluate. In the first phase, scan, a python script is used to scan domains from the Alexa Top list for QUIC support. In the second phase, collect, Chromium browser instances are used to download the domains and record their resource dependencies, and then these dependency graphs are used to download live-defended and undefended samples using the QCSD test clients. Finally in the last phase, machine learning, overhead, or shaping evaluations are performed and plots are created.

A.6 Evaluation and expected results

The main claims and associated results are described below, along with the snakemake rules used to run them in parentheses. The snakemake rules can be tested with `snakemake -j --configfile=config/test.yaml <rulename>`, where `<rulename>` is given in parentheses below. The claims can be evaluated fully using `final.yaml` instead of `test.yaml`.

- *Claim*. QCSD can successfully emulate website-fingerprinting defences such as Tamaraw and FRONT.

  Results. The Pearson correlation coefficient indicate medium and strong correlations between simulated and live-defended traces with QCSD at 50 ms sampling rates. LCSS scores indicate long common sub-sequences at 5 ms sampling rates ($> 85\%$) (`shaping_eval__all`).

- *Claim*. QCSD adds small overheads to chaff-only defences such as FRONT.

  Results. Compared to the simulated FRONT defences, the live-defended traces increase bandwidth overhead by around 30% of the original trace length and latency overhead by under 10% (`overhead_eval__table` and `overhead_eval_mconn__table`).

- *Claim*. QCSD effectively defends single connections with FRONT, but only mildly reduces classification performance when defending with Tamaraw.

  Results. In the single connection setting, the r_{20}-precision-recall curves for traces defended with FRONT match or surpass the curves of the simulated FRONT defences, whereas for the Tamaraw defence, the curves curves of the live-defended

Table 1: List of figures and snakemake rules to produce them. Use `snakemake -j --configfile=<config> <rulename>` with the appropriate rule name to create the figure. Output file paths are relative to the `results/` directory.

Section	Figure	Rule name	Output file(s)
5. Shaping Case Studies: FRONT & Tamaraw	Figure 3	`shaping_eval__all`	`plots/shaping-eval-front.png`, `plots/shaping-eval-tamaraw.png`
	Table 2	`overhead_eval__table`	`tables/overhead-eval.tex`
6.1. Defending Single Connections	Figure 4	`ml_eval_conn__all`	`plots/ml-eval-conn-tamaraw.png`, `plots/ml-eval-conn-front.png`
6.2. Defending Full Web-Page Loads	Figure 5	`ml_eval_mconn__all`	`plots/ml-eval-mconn-tamaraw.png`, `plots/ml-eval-mconn-front.png`
	Figure 6	`ml_eval_brows__all`	`plots/ml-eval-brows-front.png`
E. Overhead in the Multi-connection Setting	Table 3	`overhead_eval_mconn__table`	`tables/overhead-eval-mconn.tex`
F. Server Compliance with Shaping	Figure 8	See `failure-analysis.ipynb`	`plots/failure-rate.png`

traces do not indicate significantly better performance than undefended traces (`ml_eval_conn__all`).

- *Claim*. QCSD effectively defends multiple connections with FRONT, but does not reduce classification performance when defending with Tamaraw.

  Results. In the single connection setting, the r_{20}-precision-recall curves for traces defended with FRONT match or surpass the curves of the simulated FRONT defences, both when considering multiple orchestrated connections based on dependency graphs (`ml_eval_mconn__all`) and in the browser setting (`ml_eval_brows__all`). When applying the Tamaraw defence on multiple orchestrated connections, the precision-recall curves are similar to the curves for the undefended traces for 2 of the 3 evaluated classifiers (k-FP and VarCNN) (`ml_eval_mconn__all`).

A.7 Experiment customization

The experiments can be customised by modifying the hosts on which the experiments are to be run (e.g., `ansible/local` and `ansible/distributed`) or changing experiment parameters in the snakemake config files (e.g., `config/test.yaml` and `config/final.yaml`).

A.8 Version

Based on the LaTeX template for Artifact Evaluation V20220119.