ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

# USENIX Security '24 Artifact Appendix: NetShaper: A Differentially Private Network Side-Channel Mitigation System

Amir Sabzi, Rut Vora, Swati Goswami, Margo Seltzer, Mathias Lécuyer, Aastha Mehta
The University of British Columbia

## A Artifact Appendix

## A.1 Abstract

In this paper, we present NetShaper, a system that mitigates network side channel leaks through traffic shaping. NetShaper's traffic shaping offers differential privacy guarantees while configuring a trade-off between privacy assurances, bandwidth, and latency overheads. For evaluation, we implement NetShaper on four interconnected machines, where two of these machines serve as communicating parties, with the other two functioning as middleboxes. Furthermore, to expedite the evaluation of experiments involving multiple transmissions of traffic traces, we construct a traffic shaping simulator.

## A.2 Description & Requirements

### A.2.1 Security, privacy, and ethical concerns

Our datasets are public. The artifact does not involve destructive actions, and there is no security, privacy, or ethical concerns associated with using it.

### A.2.2 How to access

The code for both the traffic shaping simulator and our implementation of NetShaper can be found on our GitHub repository: https://github.com/ubc-systopia/netshaper/tree/AE_v2.0 . First, clone the github repository.

```
$ git clone --recursive-submodules
 ↪  https://github.com/ubc-systopia/netshaper.git
$ cd netshaper
```

To download NetShaper's dataset, change the directory to dataset directory and run the following commands:

```
$ cd dataset
$ wget
 ↪  https://zenodo.org/api/records/10783814/files-archive
$ unzip files-archive
$ tar -xf netshaper_dataset.tar.gz
```

Please note that the dataset is essential for the execution of both the simulator and the testbed implementation. Further details are available in the `README.md`.

### A.2.3 Hardware dependencies

The simulator requires a machine that should have at least 8 CPU cores, 64 GB of RAM, and a GPU with 24 GB of memory. To store the dataset and experiment results, simulator needs 100 GB of disk space. For NetShaper implementation, we use four machines interconnected in a linear topology. Each machine should have at least 8 CPU cores, 32 GB of RAM, 100 GB of disk space, and a 10 Gbps NIC. The machines should be connected to a LAN, ensuring no interference from the public network or other network applications.

### A.2.4 Software dependencies

Simulator requires `Python 3.10.6` and can be executed on arbitrary OS without root access. The list of required Python packages is available in our repository. To compile our code for NetShaper implementation, we use `gcc 11.4.0`. The implementation is specific to Ubuntu 22.04. For video server, we use Nginx 1.23.4. For transport protocol we use Microsoft implementation of the QUIC protocol, MsQUIC 2.2.4. We use our modified version of wrk2, an HTTP benchmarking tool, to send requests asynchronously. The list of main software dependencies are provided in Git repository.

### A.2.5 Benchmarks

- **Datasets:** We use two datasets, one for our video streaming service and another for our web service. Instructions for accessing these datasets can be found in Section A.2.2.
- **Models:** We use TCN model and CNN model from the Beauty and the Burst paper. Both are included in the code repository.
- **Metrics:** We report throughput as number of requests per second, latency in milliseconds, and accuracy of the classification. The relative bandwidth overhead is the number of dummy bytes transmitted normalized to the size of the unshaped traffic trace. For privacy, we report the privacy loss, $\varepsilon$, as defined in the framework of differential privacy (Equation 1).
- **Data licenses:** The Creative Commons Attribution license.

## A.3 Set-up

Here, we offer high-level steps for setting up both the simulator and our implementation. Additionally, each experiment is accompanied by dedicated setup instructions outlined in the corresponding section of the `README.md` file.

### A.3.1 NetShaper Implementation

To set up the server and client, ensure that Docker is installed on the hosting machines, clone the `sys` repository onto both of them, download and place the dataset in the `/dataset` directory, and execute the `./setup` script for the server and the client. Our scripts will handle the remaining setup procedures. To configure the middleboxes, physical access to the machines is necessary to adjust the BIOS settings and initiate a reboot. In addition to Docker, ensure that `tcpdump` is installed on these machines to facilitate the collection of traffic traces. For more details, consult the experiments' `README.md` file.

### A.3.2 Traffic Shaping Simulator

Setting up the simulator is comparatively straightforward compared to the testbed; you only need to install the necessary Python prerequisites.

```
pip install -r <path-to-simulator>/requirements.txt
```

### A.3.3 Basic Test

After downloading the dataset, to test the simulator, you can run the following commands:

```
cd evaluation/web_bandwidth
./run.sh --experiment="dp_interval_vs_overhead_web"
↪ --config_file="configs/dp_interval_vs_overhead_web.json"
```

The experiment results are stored in `reslts` directory of this particular experiment.

## A.4 Evaluation Workflow

Each experiment has its dedicated directory under the `/evaluation` directory of the NetShaper repository. To execute the experiment, run `./run.sh` with specific arguments as outlined in the experiment's `README.md` file. The parameters of the experiment controlled by a `json` file stored under `/config` directory of the experiment. Note that these parameters are specific to each experiment, and their descriptions can be found in the experiment's `README.md` file. The results of the experiments, including a `pickle file` and a plot, will be recorded in a subdirectory of the `/results` directory for the experiment, named as the experiment title along with the date and time of the execution. Certain experiments depend on a set of helper scripts for execution, data collection, and result plotting. All these scripts can be found in the `/helper` directory within the experiment.

### A.4.1 Major Claims

**(C1):** NetShaper defeats state-of-the-art classifier with privacy loss ($\varepsilon$) as large as 200. This is proven by the experiment (*E1*) described in Section 5.1 whose results are illustrated in Figure 5.

**(C2):** When shaping is disabled, the peak system throughput and the maximum concurrent clients sustained by the NetShaper middlebox align with those of a direct connection. The NetShaper middleboxes add a small latency overhead ,typically less than 10ms, when compared to a direct connection. This is proven by experiment (*E2*) described in Section 5.3. The results of these experiments are illustrated in Figure 7a and Figure 7b respectively.

**(C3):** When shaping is enabled, the latency of video segments in a video streaming application increases linearly with the DP interval. For all configurations in Section 5.4, NetShaper latency is less than $5s$. The bandwidth overhead of NetShaper decreases as the DP intervals become larger. Experiments (*E3*) and (*E4*) reproduce results illustrated in Figure 8a and Figure 8b.

**(C4):** When shaping is enabled, the latency of webpage requests in a web service increases linearly with the DP interval. The minimum bandwidth overhead occurs at $T = 50ms$. Experiments (*E5*) and (*E6*) prove the results showed in Figure 8c and Figure 8d.

**(C5):** In both video streaming and web service applications, NetShaper incurs three order of magnitude lower bandwidth overhead than constant-rate traffic shaping. The bandwidth overhead of NetShaper decreases as the number of concurrent flows increases. For video streaming and web service applications, NetShaper requires 11 flows and more than 40 flows, respectively to achieve lower overhead Pacer. Experiment (*E7*) described in Section 5.6 proves this, and the results are illustrated in Figure 9a and Figure 9b.

### A.4.2 Experiments

In this section, we provide a short description of each experiment. For more details on the experiment steps, preparation, execution, and results, please consult the dedicated `README.md` file of each experiment.

**(E1):** (Empirical Privacy)[5 human-minutes + 72 compute-hours]: In this experiment, we evaluate the accuracy of state-of-the-art classifiers under the deployment of NetShaper traffic shaping. The runtime depends on GPU architecture, and we used NVIDIA Tesla V100 to train the models. The execution scripts for this experiment are stored under the 'classifier' directory in the NetShaper repository.

- **(E2):** (Middlebox Throughput and Latency)[30 human-minutes + 4 compute-hours]: In this experiment, we measure the maximum throughput that NetShaper middleboxes can sustain, and the latency overhead imposed by NetShaper middleboxes. This experiment is located in the 'microbenchmarks' directory in the project's repository.

- **(E3):** (Video Streaming Latency Overhead)[30 human-minutes + 4 compute-hours]: This experiment evaluates the latency of video streaming application with NetShaper traffic shaping.

- **(E4):** (Video Streaming Bandwidth Overhead)[5 human-minutes + 3 compute-hours]: Measuring the bandwidth overhead of the video streaming application with NetShaper traffic shaping.

- **(E5):** (Web Service Latency Overhead)[30 human-minutes + 2 compute-hours]: In this experiment, we evaluate the latency of web service application.

- **(E6):** (Web Service Bandwidth Overhead)[5 human-minutes + 1 compute-hours]: This experiment measures the bandwidth overhead of web service with NetShaper traffic shaping.

- **(E7):** (Comparison with Related Work)[5 human-minutes + 4 compute-hours]: In this experiment, we compare bandwidth overhead of NetShaper with Pacer and Constant-rate traffic shaping for both video streaming and web service applications.

- **(E8):** (Impact of Privacy Parameters)[5 human-minutes + 1 computer-minutes] In this experiment, we evaluate the effect of the sensitivity ($\Delta_T$), noise ($\sigma_T$), and number of DP queues ($N$) on privacy loss ($\varepsilon$).

## A.5 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at `https://secartifacts.github.io/usenixsec2024/`.