USENIX Security '26 Artifact Appendix: Hop: A Modern Transport and Remote Access Protocol

Paul Flammarion
Stanford University

George Hosono*
Georgia Tech

Wilson Nguyen
Stanford University

Laura Bauman
Stanford University

Daniel Rebelsky
Stanford University

Gerry Wan
Stanford University

David Adrian
Independent

Zakir Durumeric
Stanford University

A Artifact Appendix

A.1 Abstract

We provide artifacts supporting our claims on the functionality and performance of our Hop reference implementation. Our artifact includes the full client and server source code, scripts for controlled simulation experiments, and measurement scripts used to evaluate session establishment latency, file transfer throughput, and keystroke latency. The artifact also includes the datasets and plotting scripts to reproduce all performance figures presented in the paper.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Our software requires the user to install external dependencies. Our remote communication protocol can be used in a local network as well as on the public Internet. The use of hidden mode should be preferred as a means of connection for Internet-exposed Hop servers. Do not use Hop to grant access to anyone you do not trust. The code does not execute any destructive steps, and there are no ethical concerns or risks for the evaluators.

A.2.2 How to access

The artifact is hosted at Zenodo and can be accessed at: https://doi.org/10.5281/zenodo.17953396

A.2.3 Hardware dependencies

To ensure correct building and execution of our experiments, we recommend using at least two machines (physical or virtual), each with a minimum of 2 vCPUs, 4 GB of RAM, and 8 GB of storage.

*Work conducted primarily at Stanford University.

A.2.4 Software dependencies

To run all the experiments, you will need a Linux-based machine. We suggest using Ubuntu 24.04 LTS with the following dependencies:

- `Golang v1.23`, for running and building Hop. Installation available at https://go.dev. Example installation: `apt install golang`
- `Docker v28`, for building test environments. Installation available at https://docs.docker.com
- `Python3`, for evaluation only. Installation available at https://www.python.org. Example installation: `apt install python3`
- `OpenSSH v9.7p1`, for evaluation baseline. Installation available at https://www.openssh.org. Example installation: `apt install openssh-client openssh-server`
- `Rsync v3.3`, for file transfer. Installation available at https://github.com/RsyncProject/rsync. Example installation: `apt install rsync`
- `GNU Make v3.81`, for script automation. Installation available at https://www.gnu.org/software/make. Example installation: `apt install make`
- `Mininet v2.3.0`, for the evaluation in simulation. Installation available at https://mininet.org. Example installation: `apt install mininet`
- `Java 11`, to run Typometer software only used for keystroke timing evaluation. Installation available at https://www.oracle.com/java/technologies/javase/jdk11-archive-downloads.html. Example installation: `apt install openjdk-11-jre`

To ensure proper execution of Hop, we recommend using `Go 1.23` for the project, as an issue with Hop's use of `creack/pty` may prevent a shell from opening when running on newer versions of Go.

A.2.5 Benchmarks

Our artifact includes the dataset of our SSH evaluation, serving as a benchmark for Hop evaluation under the same environment. However, since the evaluator setup will likely be different (e.g., different machines, different Internet infrastructure, different times, etc.), the artifact also provides a guide to collect this data from the evaluator's setup.

A.3 Set-up

Please be sure that the machines used for evaluation have installed all software dependencies and granted permission to listen on the port or socket specified in Hop configuration (e.g., port `77` or `7777`).

When generating credentials, make sure to specify the IP address of the remote machine and to report it correctly in the client configuration file. If DNS resolution is not available for these IP addresses, use the IP address itself as both the DNS name and the server name.

Since our experiments use SSH measurements as a baseline, you may set up a working SSH communication channel between the clients and a remote SSH server for the time to shell (E1) and the file transfer in real-world (E3), and a local SSH server for file transfer in simulation (E2) and the keystroke latency (E4) experiments.

A.3.1 Installation

1. Download the Hop artifact from https://doi.org/10.5281/zenodo.17953396 and unarchive the file if required.
2. Go to Hop's directory `cd hop-go/`
3. Fetch dependencies `go get ./...; go get -t ./...`
4. Create the default credentials to locally connect to the Docker container configuration `make cred-gen`
5. Build and start the Docker container `make serve-dev`

A.3.2 Basic Test

Be sure you have a Hop server up-and-running in the previously created Docker container.

1. Connect to the Docker container `go run hop.computer/hop/cmd/hop -C containers/client_config.toml user@127.0.0.1:7777`
2. Success is having a shell as `user` in the Docker container `example.com`.

While this strictly describes a basic Hop shell access, more configurations can be found in the artifact `README` files.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): *Time to shell:* Hop starts a non-interactive shell and exits with fewer network round-trips than SSH due to its handshake efficiency. This is proven by the experiment (E1) described in Section 8.2, whose results are reported in Figure 6.

(C2): *File Transfer Speed:* Hop claims having a functional congestion control in both simulation and real-world experiments. This is proven by the experiment (E2) and (E3) described in Section 8.2, whose results are reported in Figure 7 and Figure 8.

(C3): *Keystroke latency:* Hop shows a faster keystroke response time through an interactive shell than SSH. This is proven with the experiment (E4), which is described in Section 8.2 and shown in Figure 9.

A.4.2 Experiments

Each experiment requires a running Hop instance, and setting up a fully functional environment takes approximately 15 human-minutes per machine. As this is a one-time setup shared across all experiments, we do not include these initial 15 minutes in the reported time for each experiment below.

- Generate client and server credentials as specified in `hop-go/CONFIGURATION.md` or reuse the ones created for the Docker basic test. For (E2), create experiment-specific credentials with `hop-go/artifact/simulation/config/cred-gen-simulation.sh` script or manually.
- Start a Hop Hidden Docker container with `make serve-dev-hidden` or configure it manually to evaluate Hop Hidden mode.
- Generate file transfer files with random data by running `dd if=/dev/urandom of=100MB_file bs=1M count=100`. Reproduce for `10MB` and `1GB` files.

Most of the paths within the measurement scripts are relative to the root of the repository. You can adapt the paths to your environment. Each experiment's dataset can be found in the `hop-go/artifact` folder.

(E1): *[Time to shell] [15 human-minutes + 20 computer-minutes]:* Measures the time to perform a full handshake and a session setup with a remote machine for each protocol.
How to: Run the Python script to automatically measure connections and collect the session connection data.
Preparation: Have at least one server to which you can connect with the desired protocol (Hop Discoverable, Hop Hidden, and/or SSH). Update the config-

uration at the top of the `tts_measure.py` file in the `artifact/time-to-shell` folder.

Execution: Run the `tts_measure.py` script, which will output a `CSV` file. By default, the script runs 10 experiments. In the paper, we ran 100 measures for each of the 3 protocols on each machine. This value can also be updated at the top of the file.

Results: Visualize the results with any desired software or configure and run the `tts_plot.py` script.

(E2): *[File transfer in simulation] [15 human-minutes + 15 computer-minutes]:* This evaluates file transfer performance under controlled network conditions using Mininet. It isolates the effects of bandwidth, latency, jitter, and loss on end-to-end transfer time for Hop and SSH.

How to: Run the provided Mininet-based experiment script, which automatically sets up the network topology, performs file transfers, and records results.

Preparation: Ensure Mininet is installed and can be executed with elevated privileges, as well as Golang or Hops builds. This might require editing the sudoers file. Be sure that you have all the credentials for both Hop and SSH correctly located according to their respective configurations, including the `.hop/authorized_keys` file on Hop's server side. Experiment parameters such as queue size, bandwidth, delay, and file sizes can be adjusted at the top of the simulation script.

Execution: Execute the simulation script to initialize the Mininet topology, run the file transfer experiments for each configured network condition, and store the results in a `CSV` file.

Results: The output can be visualized using the `simulation_plot.py` script.

(E3): *[File transfer in real-world] [15 human-minutes + 2 computer-hour]:* Measures end-to-end file transfer time to a remote host in a real-world deployment. Transfers are performed for multiple file sizes (10 MB, 100 MB, and 1 GB).

How to: Run the measurement script, which performs repeated file transfers using `rsync` over SSH and Hop and records the elapsed time.

Preparation: Generate the files that you want to transfer during the experiment. Ensure that the target host is reachable and running the required SSH and/or Hop services. Configure host addresses, users, data files, and Hop configuration file paths at the top of `transfer_measure.py`.

Execution: Execute `transfer_measure.py` and monitor the file transfers to ensure their completion. On a separate execution, the TCP congestion control algorithm can be changed (e.g., Cubic to NewReno). You can change the congestion control of the machine to NewReno with `sysctl net.ipv4.tcp_congestion_control=reno`

Results: Results can be visualized using the `transfer_plot.py` script.

(E4): *[Keystroke latency] [30 human-minutes + 1 computer-hour]:* Evaluates interactive latency by measuring keystroke-to-echo delay over SSH and Hop sessions.

How to: Establish an interactive terminal session with the target host using either SSH or Hop, then use the Typometer third-party software to record keystroke latency for the active terminal window.

Preparation: Ensure that an interactive SSH or Hop session can be established with the target host. The Typometer software and its license are provided in the `keystroke-latency` directory. Follow the included Typometer 'README' to install and launch Typometer. Ensure that you are using Java 11 to start Typometer. Keep the default Typometer settings.

Execution: Start Typometer and select the active terminal window. Typometer will write a keystroke and automatically measure the time to its display. Repeat the measurement for each protocol and host under identical conditions. Export the recorded latency data as a `CSV` file.

Results: Results can be visualized using the `keystrokes_plot.py` script.

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/usenixsec2026/.