

Requirements and Software Design for a Better Tor Performance Measurement Tool

Karsten Loesing

karsten@torproject.org

Tor Tech Report

— *This report has not been published yet, so it has no number and its URL is going to change once it's published; better not reference it yet. Work in progress.* —

1 Introduction

Four years ago, we presented a simple tool to measure performance of the Tor network [1]. This tool, called Torperf, requests static files of three different sizes over the Tor network and logs timestamps of various request substeps. These data turned out to be quite useful to observe user-perceived network performance over time.¹ However, static file downloads are not the typical use case of a user browsing the web using Tor, so absolute numbers are not very meaningful. Also, Torperf consists of a bunch of shell scripts which makes it neither very user-friendly to set up and run, nor extensible to cover new use cases.

For reference, we made an earlier approach 1.5 later that suggested redesigning the Python parts in Torperf, but that redesign never happened.²

In this report we outline requirements and a software design for a rewrite of Torperf. We loosely start with non-functional requirements, so aspects like user-friendliness or extensibility, because these requirements drive the rewrite more than the immediate need for new features. After that, we discuss actual functional requirements, so experiments that we want to perform to measure things in the Tor network on a regular and automated basis. Finally we suggest a software design that fulfills all these requirements. This report is meant to be updated in the process of writing this new software, so that it can later serve as documentation. Only then it's going to be published officially.

¹<https://metrics.torproject.org/performance.html>

²<https://trac.torproject.org/projects/tor/ticket/2565>

2 Requirements

Before we talk about functional requirements, that is, actual experiments the new Torperf is supposed to perform, we want to discuss more general requirements of a Torperf rewrite. For the time being, assume that the major functional requirement is to “make client requests over the Tor network and record timestamps and other meta data for later analysis.” Whichever type of request this is, it’s important that all or most of the non-functional requirements listed in the following are met. We start with configuration requirements, followed by requirements to results formats.

2.1 Configuration requirements

2.1.1 Installation and upgrade

Whoever installs Torperf shouldn’t be required to understand its codebase, nor rely on support by someone who does.³ Ideally, the new Torperf comes as OS package, meaning that it only relies on third-party software that is also either available as OS package or that is shipped with the Torperf package. To put it simply, installing on Debian Wheezy should be as easy as `apt-get install`, and upgrading should simply be an `apt-get update && apt-get upgrade` away.

2.1.2 Run as service

The new Torperf should run as OS service, unrelated to a specific user. It should have a config file that is easy to understand, and it should come with scripts to start, restart, and stop the Torperf service. If the service operator wants to run multiple experiments at once, they’d simply configure all those experiments in the configuration file or directory, rather than starting multiple instances of Torperf.

2.1.3 Single configuration point

All requests that Torperf performs use a local Tor client, go over the Tor network, and are answered by some server. Some experiments may be designed to use remote servers not controlled by the person running Torperf, but others require setting up a custom server. In the latter case, configuring, starting, restarting, or stopping client and server should happen in a single place, that is, on a single machine as part of the Torperf service. Fortunately, running client and server on the same physical should not have any effect on measurement results, because all requests and responses traverse the Tor network.

2.1.4 User-defined tor version or binary

A key part of measurements is the tor software version or binary used to make requests or even to handle requests in the hidden-service case. It should be easy to specify a tor version that

³This sounds obvious, but this was the case with the current, shell script based Torperf, and it’s also the reason why its current known userbase is 2.

is not the current tor version shipped with the OS. Similarly, it should be easy to point to a custom tor binary to use for measurements. It should be possible to run different experiments with different tor versions or binaries in the same Torperf service instance.

2.1.5 User-defined third-party software version or binary

Similar to the previous requirement, it should be easy to specify custom versions or binaries of third-party software other than the version currently shipped with the OS. This applies, for example, to Firefox when attempting to make measurements more realistic.

2.2 Results requirements

2.2.1 Results format

The measurement results produced by Torperf contain no sensitive data by design, because all requests are made by Torperf itself, not by actual Tor users. We'll want to collect as much information as needed to perform useful analysis. But at the same time we want to keep it as easy as possible to process results.

Results may come from various sources, e.g., an HTTP/SOCKS client, Selenium/Firefox, a tor client used to make the request or to answer the request as hidden service, an HTTP server, etc. Torperf should not store original logs, because that would only shift the issue of processing different log formats to the analysis step. Such an approach would also generate unnecessarily large results files. Torperf should rather process data from these sources and store them in a custom results format that can be easily processed using tools shipped with Torperf. Results may include data which appear not immediately relevant to measuring Tor performance, but which may be useful for related purposes. For example, Torperf should include data about circuit failures in its results, even though these circuits may not have been used in actual requests.⁴ Deciding which data to store should be the responsibility of whoever designs a Torperf experiment, though formats should be somewhat uniform between experiments. A possible results data format could be JSON, but other data formats are plausible, too.

2.2.2 Results web interface

Torperf should provide all its results via a public web interface. This includes measured timestamps as well as any meta data collected in an experiment. Results should be available for all measurements as well as for a subset specified by measurement time or filtered by other criteria.

2.2.3 Results accumulator

A Torperf service instance should be able to accumulate results from its own experiments and remote Torperf service instances. Therefore, it should simply download new results from their web interfaces and incorporate them in its own database. It would be useful to have the accumulator warn if a remote service instance becomes stale and doesn't serve recent results

⁴<https://trac.torproject.org/projects/tor/ticket/8662>

anymore. Note how the service instance name or identifier should become part of Torperf results meta data.

2.2.4 Truncate old results

A Torperf service instance should be able to limit space requirements for storing past results. This could be done by discarding results that are older than a configured number of days.

2.2.5 Results graph data

Ideally, Torperf provides aggregate statistics via its web interface which can then be visualized by others. Even more ideally, Torperf serves a few HTML pages containing the necessary JavaScript to visualize results. While this may sound like going overboard here, making Torperf the tool to run performance measurements *and* to present results has the advantage of not building and maintaining a separate tool for the latter.⁵

2.2.6 Results parsing library

The new Torperf should come with an easy-to-use library to process its results. Alternatively, this library could be provided and maintained as part of stem or another parsing library.

3 Experiments

3.1 High priority experiments

3.1.1 Alexa top-X websites using Selenium and the Tor Browser

One major reason for rewriting Torperf is to make its results more realistic.⁶ It was suggested to track down Will Scott's torperf-like scripts, make them public if needed, and do a trial deployment somewhere.⁷ An experiment making these more realistic measurements should use something like Selenium to control an actual Tor Browser to make requests.⁸ As a variant, this experiment should be run with other Firefox versions, e.g. that one that uses optimistic data.⁹

⁵For reference, the current Torperf produces measurement results which are re-formatted by metrics-db and visualized by metrics-web with help of metrics-lib. Any change to Torperf triggers subsequent changes to the other three codebases, which is suboptimal.

⁶<https://trac.torproject.org/projects/tor/ticket/7168>

⁷<https://trac.torproject.org/projects/tor/ticket/7516>

⁸Related to this, Zack Weinberg has written code that downloads the present Tor Browser alpha and Python Selenium client and monkey-patches them to play nice together; see <https://github.com/zackw/tailwagger>. This code is very specialized for his requirements right now, but might still be of use to us. We should look especially at what is done to TBB and python-selenium by the patches under pkg/tor-browser-3.0a3 and pkg/python-selenium-2.33.0, and the TbbDriver class in client/tbbselenium.py.

⁹<https://trac.torproject.org/projects/tor/ticket/3875>

3.1.2 Static file downloads

Another major reason for rewriting Torperf is to supersede the existing, bit-rotting codebase. The new Torperf should therefore provide an experiment that is identical to the current, single Torperf experiment: download static files of sizes 50 KiB, 1 MiB, and 5 MiB over the Tor network and record timestamps and relevant meta data.

There are quite a few details in getting these downloads right, which shall not all be specified here. For example, Torperf needs to enforce using a fresh circuit for each run, which is currently ensured by reducing the maximum circuit dirtiness to a value that is lower than the experiment period. An alternative may be to send the NEWNYM signal to the tor process after every stream.¹⁰ The easiest path might be to leverage on the stream isolation features introduced in tor 0.2.3¹¹ and use a different SOCKS username/password for each requests.

The best way to extract these requirements is to read the source.¹² There also exists a proof-of-concept implementation of this experiment in Twisted which can and should serve as starting point for this rewrite.¹³

The results format of the current Torperf codebase shall serve as blueprint for designing a new results format.¹⁴ As stated before, the new results format should make use of JSON or another data format, so that results can be processed more easily.

Results may even contain more information than the original Torperf results. For example, assuming that the new Torperf service controls both HTTP client and server, new timestamps could be added for serving the first and last byte. Matching timestamps between client and server can be achieved by serving new random content in each request and use this content (or a digest thereof) as request identifier.

3.2 Lower priority experiments

3.2.1 Canonical median web page

A lower-priority experiment would be to devise and deploy the canonical median web page.¹⁵ Such an experiment would share a lot with static file downloads, but would serve an actual web page using Torperf's own webserver.

3.2.2 Hidden service performance

Another possible experiment would be to measure performance to a hidden service.¹⁶ This hidden service could easily run on the same tor client instance, or on a separate tor client instance running on the same physical host. We'll want to add additional timestamps for hidden service events.¹⁷ The hidden service could serve static files, a canonical median web page, or

¹⁰<https://trac.torproject.org/projects/tor/ticket/2766>

¹¹<https://gitweb.torproject.org/torspec.git/blob/HEAD:/proposals/171-separate-streams.txt#193>

¹²<https://gitweb.torproject.org/torperf.git/tree>

¹³<https://gitweb.torproject.org/karsten/torperf.git/tree/refs/heads/perfd>

¹⁴<https://metrics.torproject.org/formats.html#torperf>

¹⁵<https://trac.torproject.org/projects/tor/ticket/7517>

¹⁶<https://trac.torproject.org/projects/tor/ticket/1944>

¹⁷<https://trac.torproject.org/projects/tor/ticket/2554>

even an Alexa top-X website.

The latter has the minor disadvantage that server timestamps cannot be matched with client timestamps based on content, but only probabilistically via timing. This may become problematic when running lots and lots of experiments at the same time. Or maybe it's possible to match client and server observations using identifiers, like the rendezvous cookie, exchanged in the rendezvous process instead.

3.2.3 GET POST performance

The experiments so far all measured download speed, but it's also easy to measure upload speed.¹⁸ Torperf's own web server could accept POST data and measure timestamp of first byte received, last byte received, etc. Similar to static file downloads, client and server timestamps can be matched based on the random data that is posted to the server, or a digest of that data.

Ideally such upload tests should also be done using a full web browser driven by Selenium.

4 Software design

The previous sections outlined requirements to the Torperf rewrite and possible experiments, unrelated to an actual implementation. The purpose of this section is to suggest a possible software design matching these requirements. It's quite possible that there are better software designs. This section should serve as starting point for a discussion.

We have an initial proof-of-concept prototype that uses Twisted to implement a small subset of the new Torperf. Even though the new Torperf does quite a few things at once, like starting tor processes, making requests, answering requests, collecting results, etc., it can be implemented as a single Twisted application. This shifts some deployment problems to what people usually do when deploying Twisted applications, rather than forcing us to solve these problems yet another time. Of course, at the same time it forces us to follow the Twisted model of writing applications, which seems not too bad in this case. The current prototype also requires twisted-socks as SOCKS client, txtorcon for communicating with tor clients, and stem for event parsing.

The following list outlines tasks that the new Torperf needs to perform. These could be implemented as Python classes, though this list was not written with Python or Twisted specifics in mind:

configuration handler Validate and parse configuration file or directory, provide configuration values to other application parts.

logger Log operation details for debugging purposes and for normal service operation, unrelated to storing measurement results.

tor process starter Configure, start, hup, and stop local tor client processes, using a previously configured tor version or binary.

¹⁸<https://trac.torproject.org/projects/tor/ticket/7010>

tor controller Connect to tor's control port, force-set guards if required by the experiment, register for and handle asynchronous events, set up and tear down hidden services.

request scheduler Start new requests following a previously configured schedule.

request runner Handle a single request from creation over various possible sub states to timeout, failure, or completion.

HTTP/SOCKS client Make HTTP GET requests using our own SOCKS client and connecting to a local tor client to gather as many timestamps of substeps as possible.

Selenium/Firefox wrapper Make web request using Selenium/Firefox, possibly using Xvfb and using a previously configured Firefox version or binary, and collect as many timestamps of substeps as possible.

SOCKS or HTTP proxy Capture even more timestamps by proxying requests made via Selenium/Firefox through our own SOCKS or HTTP proxy before handing them to the local tor client.

HTTP server Run on port 80, serve static files and the canonical median web page, accept POST requests, provide measurement results via RESTful API, present measurement results on web page.

Alexa top-X web pages updater Periodically retrieve list of top-X web pages.

results database Store request details, retrieve results, periodically delete old results if configured.

results accumulator Periodically collect results from other Torperf instances, warn if they're out of date.

analysis scripts Command-line tools to process measurement results and produce graphs and other aggregate statistics. Could also become part of stem instead, together with a tutorial.

References

- [1] Karsten Loesing. Performance of requests over the Tor network. Technical Report 2009-09-001, The Tor Project, September 2009.