simple-bw-scanner Documentation

Release get_versions()['version']

['Matt Traudt [sirmatt at ksu dot edu], juga [juga at riseup dot net]

Mar 16, 2021

Contents

CHAPTER 1

User main documentation

Included in the repository root and in `sbws` Debian package:

1.1 Readme

Build Status

Simple Bandwidth Scanner (called `sbws`) is a Tor bandwidth scanner that generates bandwidth files to be used by Directory Authorities.

The scanner measures the bandwidth of each relay in the Tor network (except the directory authorities) by creating a two hops circuit with the relay. It then measures the bandwidth by downloading data from a destination Web Server and stores the measurements.

The generator read the measurements, aggregates, filters and scales them using torflow's scaling method. Then it generates a bandwidth list file that is read by a directory authority to report relays' bandwidth in its vote.

WARNING: This software is intended to be run by researchers using a test Tor network, such as chutney or shadow, or by the Tor bandwidth authorities on the public Tor network. Please do not run this software on the public Tor network unless you are one of the Tor bandwidth authorities, to avoid creating unnecessary traffic.

ADVICE: It is recommended to read this documentation at Read the Docs. In Tor Project Gitlab (tpo Gitlab) some links won't be properly rendered. It can also be read after installing the Debian package `sbws-doc` in `/usr/share/doc/sbws` or after building it locally as explained in `./docs/source/documenting.rst`.

1.1.1 Installing

See ./INSTALL.rst (in local directory or tpo Gitlab) or INSTALL.html (local build or Read the Docs).

1.1.2 Deploying and running

See ./DEPLOY.rst (in local directory or tpo Gitlab) or DEPLOY.html (local build or Read the Docs).

1.1.3 Changelog

See ./CHANGELOG.rst (in local directory or tpo Gitlab) or CHANGELOG.html (local build or Read the Docs).

1.1.4 Documentation

More extensive documentation can be found in the `./docs` directory, and online at sbws.rtfd.io.

1.1.5 License

1.1.6 Authors

See *./AUTHORS.md* (in local directory or tpo Gitlab) or AUTHORS.html (local build or Read the Docs).

1.2 Installing Simple Bandwidth Scanner

The recommended method is to install it from your system package manager.

In Debian/Ubuntu systems:

```
sudo apt install sbws
```

To install also the documentation:

```
sudo apt install sbws-doc
```

You might need to check in which releases is the package available.

There is a port for FreeBSD.

Continue reading to install `sbws` in other ways.

1.2.1 System requirements

- Tor (last stable version is recommended)
- Python 3 (>= 3.6)

1.2.2 Python dependencies

- Stem >= 1.7.0
- Requests (with socks support) >= 2.10.0

It is recommend to install the dependencies from your system package manager. If that is not possible, because the Python dependencies are not available in your system, you can install them from their sources. We only recommend using pip for development or testing.

1.2.3 Installing sbws from source

Clone `sbws`:

```
git clone https://git.torproject.org/sbws.git
git checkout maint-1.1
```

The branch `maint-1.1` is the last stable version and the one that should be used in production.

and install it:

```
cd sbws
python3 setup.py install
```

1.2.4 Installing sbws for development or testing

If you use pip, it is recommended to use virtualenv, to avoid having different versions of the same libraries in your system.

To create a `virtualenv`:

```
virtualenv venv -p /usr/bin/python3
source venv/bin/activate
```

Clone `sbws`:

```
git clone https://git.torproject.org/sbws.git
```

Install the python dependencies:

```
cd sbws && pip install -e .
```

1.2.5 Configuration and deployment

`sbws` needs *destination* s to request files from.

Please, see DEPLOY.rst (in the local directory or Tor Project Gitlab) or DEPLOY.html (local build or Read the Docs) to configure, deploy and run `sbws`.

1.2.6 System physical requirements

- Bandwidth: at least 12.5MB/s (100 Mbit/s).
- Free RAM: at least 2GB
- Free disk: at least 3GB

`sbws` and its dependencies need around 20MB of disk space. After 57 days `sbws` data files use a maximum of 3GB. If `sbws` is configured to log to files (by default will log to the system log), it will need a maximum of 500MB.

It is recommended to set up an automatic disk space monitoring on `sbws` data and log partitions.

Details about `sbws` data:

`sbws` produces around 100MB of data a day. By default raw results' files are compressed after 29 days and deleted after 57. The bandwidth files are compressed after 7 days and deleted after 1. After 57 days, the disk space used by

the data will be up to 3GB. It will not increase further. If `sbws` is configured to log to files, logs will be rotated after they are 10MB and it will keep 50 rotated log files.

1.3 Deploying Simple Bandwidth Scanner

To run sbws is needed:

- A machine to run the *scanner*.
- One or more *destination* (s) that serve a large file.

Both the `scanner` and your the `destination` (s) should be on fast, well connected machines.

1.3.1 destination requirements

- A Web server installed and running that supports HTTP GET, HEAD and Range (**RFC 7233**) requests. `Apache` HTTP Server and `Nginx` support them.
- TLS support to avoid HTTP content caches at the various exit nodes.
- Certificates can be self-signed.
- A large file; at the time of writing, at least 1 GiB in size It can be created running:

```
head -c $((1024*1024*1024)) /dev/urandom > 1GiB
```

- A fixed IP address or a domain name.
- Bandwidth: at least 12.5MB/s (100 Mbit/s).
- Network traffic: around 12-15GB/day.

If possible, use a Content delivery network (CDN) in order to make the destination IP closer to the scanner exit.

1.3.2 scanner setup

Install sbws according to INSTALL.rst (in the local directory or Tor Project Gitlab) or INSTALL.html (local build or Read the Docs).

To run the `scanner` it is mandatory to create a configuration file with at least one `destination`. It is recommended to set several `destinations` so that the `scanner` can continue if one fails.

If `sbws` is installed from the Debian package, then create the configuration file in `/etc/sbws/sbws.ini`. You can see an example with all the possible options here, note that you don't need to include all of that and that everything that starts with `#` and `;` is a comment:

Listing 1.1: Example sbws.example.ini

```
# Minimum configuration that needs to be customized
[scanner]
# A human-readable string with chars in a-zA-Z0-9 to identify your scanner
nickname = sbws_default
# ISO 3166-1 alpha-2 country code where the Web server destination is located.
# Default AA, to detect it was not edited.
country = SN

[destinations]
```

(continues on next page)

(continued from previous page)

```
# With several destinations, the scanner can continue even if some of them
# fail, which can be caused by a network problem on their side.
# If all of them fail, the scanner will stop, which
# will happen if there is network problem on the scanner side.

# A destination can be disabled changing `on` by `off`
foo = on

[destinations.foo]
# the domain and path to the 1GB file.
url = https://example.com/does/not/exist.bin
# Whether to verify or not the TLS certificate. Default True
verify = False
# ISO 3166-1 alpha-2 country code where the Web server destination is located.
# Default AA, to detect it was not edited.
# Use ZZ if the location is unknown (for instance, a CDN).
country = ZZ

# Number of consecutive times that a destination could not be used to measure
# before stopping to try to use it for a while that by default is 3h.
max_num_failures = 3

## The following logging options are set by default.
## There is no need to change them unless other options are prefered.
; [logging]
; # Whether or not to log to a rotating file the directory paths.log_dname
; to_file = yes
; # Whether or not to log to stdout
; to_stdout = yes
; # Whether or not to log to syslog
; # NOTE that when sbws is launched by systemd, stdout goes to journal and
; # syslog.
; to_syslog = no

; # Level to log at. Debug, info, warning, error, critical.
; # `level` must be set to the lower of all the handler levels.
; level = debug
; to_file_level = debug
; to_stdout_level = info
; to_syslog_level = info
; # Format string to use when logging
; format = %(module)s[%(process)s]: <%(levelname)s> %(message)s
; # verbose formatter useful for debugging
; to_file_format = %(asctime)s %(levelname)s %(threadName)s %(filename)s:%(lineno)s -
↪%(funcName)s - %(message)s
; # Not adding %(asctime)s to to stdout since it'll go to syslog when using
; # systemd, and it'll have already the date.
; to_stdout_format = ${format}
; to_syslog_format = ${format}

# To disable certificate validation, uncomment the following
# verify = False
```

If `sbws` is installed from the sources as a non-root user then create the configuration file in `~/.sbws.ini`.

More details about the configuration file can be found in `./docs/source/man_sbws.ini.rst` (in the local directory or Tor Project Gitlab) or man_sbws.ini.html (local build or Read the Docs) or `man sbws.ini` (system

package).

See also `./docs/source/man_sbws.rst` (in the local directory or Tor Project Gitlab) or man_sbws.html (local build or Read the Docs) or `man sbws` (system package).

1.4 Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

1.4.1 v1.1.0 (2019-03-27)

New

- V3bwfile: Report excluded relays. Closes: #28565.
- V3bwfile: Add time to report half network. Closes: #28983
- Destination: Recover destination when it failed. Closes: #29589.
- V3bwfile: Report relays that fail to be measured. Closes: #28567.
- V3bwfile: Report relays that are not measured measured. Closes: #28566
- V3bwfile: Add KeyValues to monitor relays. Closes: #29591.
- Docs: document that authorities are not measured. Closes: #29722
- Scanner: Warn when there is no progress. Closes: #28652

Fix

- v3bwfile: Report relays even when they don't reach a minimum number. Closes: #29853.
- Minor fixes. Closes #29891.
- Relaylist: Convert consensus bandwidth to bytes.

1.4.2 v1.0.5 (2019-03-06)

- Release v1.0.5. this time with the correct version

1.4.3 v1.0.4 (2019-03-06)

- Release v1.0.4. because there was a commit missing between *1.0.3* and *1.0.4-dev0* and what is released as *1.0.3* has version *1.0.4-dev0* and it can not be fixed now.

1.4.4 v1.0.3 (2019-02-28)

Fixed

- scanner: check that ResultDump queue is not full Fixes bug #28866. Bugfix v0.1.0.
- config: set stdout log level to cli argument. Closes: #29199
- cleanup: Use getpath to get configuration paths. Bugfix v0.7.0.
- destination: stop running twice usability tests. Fixes bug #28897. Bugfix v0.3.0
- globals, stem: explain where torrc options are. Fixes bug #28646. Bugfix v0.4.0
- stem: disable pad connections. Fixes bug 28692. Bugfix v0.4.0
- generate: Load all results, including error ones. Closes #29568. Bugfix v0.4.0 (line introduced in v0.1.0).
- relayprioritizer: Stop prioritizing relays that tend to fail. Fixes bug #28868. Bugfix v0.1.0
- circuitbuilder: Stop building the circuit 3 times. Fixes bug #29295. Bugfix v0.1.0.
- docs: add verify option to man and example. Closes bug #28788. Bugfix v0.4.0.
- CI: run scanner using the test network. Fixes bug #28933. Bugfix v0.1.0.
- scanner: catch SIGINT in the main loop. Fixes bug #28869. Bugfix v0.1.0.
- Stop including tests network as binary blob. Fixes bug #28590. Bugfix v0.4.0.
- relaylist: remove assertions that fail measurement. Closes #28870. Bugfix v0.4.0
- config: Use configuration provided as argument. Fixes bug #28724. Bugfix v0.7.0.
- stem: parse torrc options that are only a key. Fixes bug #28715. Bugfix v0.1.1
- stem: Stop merging multiple torrc options with the same name. Fixes bug #28738. Bugfix v0.1.1
- docs: add note about syslog when running systemd. Closes bug #28761. Bugfix v0.6.0
- CI: include deb.torproject.org key. Closes #28922. Bugfix v1.0.3-dev0
- config: stop allowing http servers without tls. Fixes bug #28789. Bugfix v0.2.0.
- Make info level logs more clear and consistent. Closes bug #28736. Bugfix v0.3.0.
- CI: check broken links in the docs. Closes #28670.
- docs: add scanner and destination requirements. Closes bug #28647. Bugfix v0.4.0
- generate: use round_digs variable name in methods. Closes bug #28602. Bugfix 1.0.3-dev0
- docs: Change old broken links in the documentation. Closes #28662.
- docs: replace http by https in links. Closes #28661.
- Fix git repository link. Fixes bug #28762. Bugfix v1.0.0.
- docs: add example destination in DEPLOY. Closes #28649.
- docs: Change links to be interpreted by ReST. Closes #28648.
- Force rtfd.io to install the package. Closes bug #28601.
- config: continue when the file is not found. Closes: #28550.
- Stop resolving domains locally and check same flags for the 2nd hop. Closes bug #28458, #28471. Bugfix 1.0.4.
- Limit the relays' bandwidth to their consensus bandwidth. Closes #28598.

- globals: add torrc logging options. Closes #28645. Bugfix v0.2.0.
- Limit bandwidth to the relay MaxAdvertisedBandwidth Fixes bug #28588. Bugfix 0.8.0.
- Exclude results, then check for the minimum number. Closes bug 28572.
- Make sbws round to 3 significant figures in torflow rounding mode. Bugfix on 27337 in sbws 1.0. Part of 28442.

Changed

- tests: remove unused testnets. Fixes bug #29046. Bugfix v0.4.0.
- scanner, destination: Log all possible exceptions.
- docs: Update/improve documentation on how the scanner/generator work. Closes: #29149
- Requests: Change make_session to use the TimedSession.
- CI: change to Ubuntu Xenial.
- docs: stop editing changelog on every bug/ticket. Closes ticket #28572.
- Change sbws scaling method to torflow. Closes: #28446.
- Round bandwidths to 2 significant digits by default. Implements part of proposal 276. Implements 28451.

Added

- Send scanner metadata as part of every HTTP request. Closes: #28741
- scanner: log backtrace when not progressing. Closes: 28932

1.4.5 v1.0.2 (2018-11-10)

Fixed

- Update bandwidth file specification version in the `generator` (#28366).
- Use 5 "=" characters as terminator in the bandwidth files (#28379)

Changed

- Include the headers about eligible relays in all the bandwidth files, not only in the ones that does not have enough eligible relays (#28365).

1.4.6 v1.0.1 (2018-11-01)

Changed

- Change default directories when sbws is run from a system service (#28268).

1.4.7 v1.0.0 (2018-10-29)

Important changes:

- `generate` includes extra statistics header lines when the number of eligible relays to include is less than the 60% of the network. It does not include the relays' lines.
- Speed up `scanner` by disabling RTT measurements and waiting for measurement threads before prioritizing again the list of relays to measure.

Fixed

- Update python minimal version in setup (#28043)
- Catch unhandled exception when we fail to resolve a domain name (#28141)
- Bandwidth filtered is the maximum between the bandwidth measurements and their mean, not the minimum (#28215)
- Stop measuring the same relay by two threads(#28061)

Changed

- Move `examples/` to `docs/` (#28040)
- Number of results comparison and number of results away from each other are incorrect (#28041)
- Stop removing results that are not away from some other X secs (#28103)
- Use secs-away when provided instead of data_period (#28105)
- Disable measuring RTTs (#28159)
- Rename bandwidth file keyvalues (#28197)

1.4.8 Added

- Write bw file only when the percentage of measured relays is bigger than 60% (#28062)
- When the percentage of measured relays is less than the 60%, do not include the relays in the bandwidth file and instead include some statistics in the header (#28076)
- When the percentage of measured relays is less than the 60% and it was more before, warn about it (#28155)
- When the difference between the total consensus bandwidth and the total in the bandwidth lines is larger than 50%, warn (#28216)
- Add documentation about how the bandwidth measurements are selected and scaled before writing them to the Bandwidth File (#27692)

1.4.9 v0.8.0 (2018-10-08)

Important changes:

- Implement Torflow scaling/aggregation to be able to substitute Torflow with sbws without affecting the bandwidth files results.
- Change stem dependency to 1.7.0, which removes the need for `dependency_links`

- Update and cleanup documentation

Added

- Add system physical requirements section to INSTALL (#26937)
- Warn when there is not enough disk space (#26937)
- Implement Torflow scaling (#27108)
- Create methods to easy graph generation and obtain statistics to compare with current torflow results.(#27688)
- Implement rounding bw in bandwidth files to 2 insignificant digits(#27337)
- Filter results in order to include relays in the bandwidth file that:(#27338)
- have at least two measured bandwidths
- the measured bandwidths are within 24 hours of each other
- have at least two descriptor observed bandwidths
- the descriptor observed bandwidths are within 24 hours of each other

Fixed

- Broken environment variable in default sbws config. To use envvar $FOO, write $$FOO in the config.
- Stop using directory as argument in integration tests (#27342)
- Fix typo getting configuration option to allow logging to file (#27960)
- Set int type to new arguments that otherwise would be string (#27918)
- Stop printing arguments default values, since they are printed by default (#27916)
- Use dash instead of underscore in new cli argument names (#27917)

Changed

- sbws install doc is confusing (#27341)
- Include system and Python dependencies in `INSTALL`.
- Include dependencies for docs and tests in `INSTALL`.
- Point to `DEPLOY` to run sbws.
- Remove obsolete sections in `INSTALL`
- Simplify `DEPLOY`, reuse terms in the `glossary`.
- Remove obsolete `sbws init` from `DEPLOY`.
- Point to config documentation.
- Add, unify and reuse terms in `glossary`.
- refactor v3bwfile (#27386): move scaling method inside class
- use custom `install_command` to test installation commands while `dependency_links` is needed until #26914 is fixed. (#27704)
- documentation cleanup (#27773)

- split, merge, simplify, extend, reorganize sections and files
- generate scales as Torflow by default (#27976)
- Replace stem `dependency_links` by stem 1.7.0 (#27705). This also eliminates the need for custom `install_command` in tox.

1.4.10 v0.7.0 (2018-08-09)

Important changes:

- `cleanup/stale_days` is renamed to `cleanup/data_files_compress_after_days`
- `cleanup/rotten_days` is renamed to `cleanup/data_files_delete_after_days`
- sbws now takes as an argument the path to a config file (which contains `sbws_home`) instead of `sbws_home` (which contains the path to a config file)

Added

- Log line on start up with sbws version, platform info, and library versions (trac#26751)
- Manual pages (#26926)

Fixed

- Stop deleting the latest.v3bw symlink. Instead, do an atomic rename. (#26740)
- State file for storing the last time `sbws scanner` was started, and able to be used for storing many other types of state in the future. (GH#166)
- Log files weren't rotating. Now they are. (#26881)

Changed

- Remove test data v3bw file and generate it from the same test. (#26736)
- Stop using food terms for cleanup-related config options
- cleanup command now cleans up old v3bw files too (#26701)
- Make sbws more compatible with system packages: (#26862)
- Allow a configuration file argument
- Remove directory argument
- Create minimal user configuration when running
- Do not require to run a command to initialize
- Initialize directories when running
- Do not require configuration file inside directories specified by the configuration

1.4.11 v0.6.0 (2018-07-11)

Important changes:

- The way users configure logging has changed. No longer are most users expected to be familiar with how to configure python's standard logging library with a config file. Instead we've abstracted out the setting of log level, format, and destinations to make these settings more accessible to users. Expert users familiar with the logging config file format can still make tweaks.

Summary of changes:

- Make logging configuration easier for the user.
- Add UML diagrams to documentation. They can be found in docs/source/images/ and regenerated with `make umlsvg` in docs/.

Added

- UML diagrams to documentation. In docs/ run `make umlsvg` to rebuild them. Requires graphviz to be installed.(GHPR#226)
- Add metadata to setup.py, useful for source/binary distributions.
- Add possibility to log to system log. (#26683)
- Add option to cleanup v3bw files. (#26701)

Fixed

- Measure relays that have both Exit and BadExit as non-exits, which is how clients would use them. (GH#217)
- Could not init sbws because of a catch-22 related to logging configuration. Overhaul how logging is configured. (GH#186 GHPR#224)
- Call write method of V3BWFile class from the object instance. (#26671)
- Stop calculating median on empty list .(#26666)

Changed

- Remove is_controller_ok. Instead catch possible controller exceptions and log them

Removed

- Two parsing/plotting scripts in scripts/tools/ that can now be found at https://github.com/pastly/v3bw-tools

1.4.12 v0.5.0 (2018-06-26)

Important changes:

- Result format changed, causing a version bump to 4. Updating sbws to 0.5.0 will cause it to ignore results with version less than 4.

Summary of changes:

- Keep previously-generated v3bw files

- Allow a relay to limit its weight based on RelayBandwidthRate/MaxAdvertisedBandwidth
- 1 CPU usage optimization
- 1 memory usage optimization

Added

- Use a relay's {,Relay}BandwidthRate/MaxAdvertisedBandwidth as an upper bound on the measurements we make for it. (GH#155)
- Ability to only consider results for a given relay valid if they came from when that relay is using its most recent known IP address. Thanks Juga. (GH#154 GHPR#199)
- Maintenance script to help us find functions that are (probably) no longer being called.
- Integration test(s) for RelayPrioritizer (GHPR#206)
- Git/GitHub usage guidelines to CONTRIBUTING document (GH#208 GHPR#215)

Fixed

- Make relay priority calculations take only ~5% of the time they used to (3s vs 60s) by using sets instead of lists when selecting non-Authority relays. (GH#204)
- Make relay list refreshing take much less time by not allowing worker threads to dogpile on the CPU. Before they would all start requesting descriptors from Tor at roughly the same time, causing us to overload our CPU core and make the process take unnecessarily long. Now we let one thread do the work so it can peg the CPU on its own and get the refresh done ASAP. (GH#205)
- Catch a JSON decode exception on malformed results so sbws can continue gracefully (GH#210 GHPR#212)

Changed

- Change the path where the Bandwidth List files are generated: now they are stored in `v3bw` directory, named `YYmmdd_HHMMSS.v3bw`, and previously generated ones are kept. A `latest.v3bw` symlink is updated. (GH#179 GHPR#190)
- Code refactoring in the v3bw classes and generation area
- Replace v3bw-into-xy bash script with python script to handle a more complex v3bw file format (GH#182)

1.4.13 v0.4.1 (2018-06-14)

Changed

- If the relay to measure is an exit, put it in the exit position and choose a non-exit to help. Previously the relay to measure would always be the first hop. (GH#181)
- Try harder to find a relay to help measure the target relay with two changes. Essentially: (1) Instead of only picking from relays that are 1.25 - 2.00 times faster than it by consensus weight, try (in order) to find a relay that is at least 2.00, 1.75, 1.50, 1.25, or v1.00 times as fast. If that fails, instead of giving up, (2) pick the fastest relay in the network instead of giving up. This compliments the previous change about measuring target exits in the exit position.

Fixed

- Exception that causes sbws to fall back to one measurement thread. We first tried fixing something in this area with `88fae60bc` but neglected to remember that `.join()` wants only string arguments and can't handle a `None`. So fix that.
- Exception when failing to get a relay's `ed25519_master_key` from Tor and trying to do `.rstrip()` on a None.
- `earliest_bandwidth` being the newest bw not the oldest (thanks juga0)
- `node_id` was missing the character "$" at the beginning

1.5 Authors

The following people have contributed to Simple Bandwidth Scanner. Thank you for helping make Tor better.

- anadahz
- George Kadianakis
- Georg Koppen
- juga
- Matt Traudt
- teor

Last updated: 2020-06-26 on d7a822bf

1.6 Simple Bandwidth Scanner - SBWS(1)

1.6.1 SYNOPSIS

sbws [**Optional arguments**] [**Positional arguments**]

sbws [**-h**] [**–version**] [**–log-level** {**debug,info,warning,error,critical**}] [**-c** CONFIG] {**cleanup,scanner,generate,init,stats**}

1.6.2 DESCRIPTION

Tor bandwidth scanner that generates bandwidth measurements files to be read by the Directory Authorities.

The **scanner** requires a configuration file (see **sbws.ini** (5)) with a with a '[destinations]' section.

sbws can be run a python script or a system service. The later is recommended.

The default locations of the files that **sbws** reads or generate depend on on how it is run. See the section **FILES** to know which are the default locations.

1.6.3 OPTIONS

Positional arguments

{**cleanup,scanner,generate,init,stats**}

These arguments can have additional optional arguments. To obtain information about them, run: 'sbws <positional argument> –help'.

Optional arguments

-h, --help Show help message and exit.

--version Show **sbws** version and exit.

–log-level {debug,info,warning,error,critical} Override the sbws log level (default: info).

-c CONFIG, --config CONFIG Path to a custom configuration file.

1.6.4 EXAMPLES

sbws scanner Run the scanner using **sbws** defaults.

sbws -c ~/.sbwsrc scanner Run the scanner using the configuration file in *~/.sbwsrc*

sbws –log-level debug generate Generate v3bw file in the default v3bw directory.

sbws cleanup Cleanup datadir and v3bw files older than XX in the default v3bw directory.

1.6.5 FILES

In the following list, the first path is the default location when running **sbws** as an script, the second path is the default location when running **sbws** as a system service.

$HOME/.sbws.ini or /etc/sbws/sbws.ini Location where **sbws** searchs for a custom configuration file, when the option **–config** is not provided.

$HOME/.sbws or /var/lib/sbws Location where **sbws** writes/reads measurement data files, bandwidth list files and **tor** process data.

Under this directory, **sbws** creates the following subdirectories:

datadir Raw results generated by the `sbws scanner`. Other commands (such as `generate` and `stats`) read results from this directory.

log Log files generated by `sbws`, when logging to a file is configured (see **sbws.ini**).

v3bw Bandwidth files generated by `sbws generate`. These are the files read by the Tor directory authorities.

tor Data generated by the **tor** process launched by **sbws**.

$HOME/.sbws/tor or /run/sbws/tor Location where the **tor** process launched by `sbws scanner` stores temporal files, like Unix domain sockets.

1.6.6 SEE ALSO

sbws.ini (5), https://sbws.readthedocs.org, https://gitweb.torproject.org/torspec.git/tree/bandwidth-file-spec.txt, **tor** (1).

1.6.7 BUGS

Please report bugs at https://gitlab.torproject.org/tpo/network-health/sbws/-/issues/.

1.7 Simple Bandwidth Scanner - SBWS.INI(5)

1.7.1 DESCRIPTION

Tor bandwidth scanner configuration file.

sbws (1) `scanner` command requires a configuration file with the "[scanner]", "[destinations]" "[destination.<name>]" sections.

There must be at least one "[destination.<name>]".

See an **EXAMPLES** below for a minimal configuration.

1.7.2 SECTIONS

general

data_period = INT Days into the past that measurements are considered valid. (Default: 5)

http_timeout = INT Timeout in seconds to give to the python Requests library. (Default: 10)

circuit_timeout = INT Timeout in seconds to create circuits. (Default: 60)

reset_bw_ipv4_changes = {on, off} Whether or not to reset the bandwidth measurements when the relay's IP address changes. If it changes, we only consider results for the relay that we obtained while the relay was located at its most recent IP address. (Default: off)

reset_bw_ipv6_changes = off NOT implemented for IPv6.

paths

When **sbws** is run as a system service, *~/.sbws* is changed to */var/lib/sbws*.

sbws_home = STR sbws home directory. (Default: ~/.sbws)

datadir = STR Directory where sbws stores temporal bandwidth results files. (Default: ~/.sbws/datadir)

v3bw_dname = STR Directory where sbws stores the bandwidth list files. These are the files to be read by the Tor Directory Authorities. (Default: ~/.sbws/v3bw)

v3bw_fname = STR File names of the bandwidth list files. The latest bandwidth file is symlinked by `latest.v3bw`

state_fname = STR File path to store the timestamp when the scanner was last started. (Default: ~/.sbws/state.dat)

log_dname = STR Directory where to store log files when logging to files is enabled. (Default: ~/.sbws/log)

destinations

It is required to set at least one destination for the scanner to run. It is recommended to set several destinations so that the scanner can continue if one fails.

STR = {on, off} Name of destination. It is a name for the Web server from where to download files in order to measure bandwidths.

usability_test_interval = INT How often to check if a destination is usable (Default: 300)

destinations.STR

url = STR The URL to the destination. It must include a file path. It can use both http or https.

verify = BOOL Whether or not to verify the destination certificate. (Default: True)

country = STR ISO 3166-1 alpha-2 country code. Use ZZ if the destination URL is a domain name and it is in a CDN.

tor

When **sbws** is run as a system service *~/.sbws/tor* is replaced by */run/sbws/tor*.

datadir = STR sbws' owned tor directory. (Default: ~/.sbws/tor)

control_socket = STR sbws's owned tor control socket file. (Default: ~/.sbws/tor/sbws/control)

pid = STR sbws's owned tor pid file. (Default: ~/.sbws/tor/sbws/tor.pid)

log = STR sbws's owned tor directory log files. (Default: ~/.sbws/tor/log)

external_control_port = INT tor control port to connect to. Useful to run integration tests with chutney. (Default: not set. If set, it takes preference over the control socket)

extra_lines = sbws's tor extra configuration. (Default: None)

scanner

nickname = STR A human-readable string with chars in a-zA-Z0-9 to identify the scanner. (Default: IDidntEditTheSBWSConfig)

country = STR ISO 3166-1 alpha-2 country code. (Default: AA, a non existing country to detect it was not edited)

download_toofast = INT Limits on what download times are too fast/slow/etc. (Default: 1)

download_min = INT Limits on what download times are too fast/slow/etc. (Default: 5)

download_target = INT Limits on what download times are too fast/slow/etc. (Default: 6)

download_max = INT Limits on what download times are too fast/slow/etc. (Default: 10)

num_rtts = INT How many RTT measurements to make. (Default: 0)

num_downloads = INT Number of downloads with acceptable times we must have for a relay before moving on. (Default: 5)

initial_read_request = INT The number of bytes to initially request from the server. (Default: 16384)

measurement_threads = INT How many measurements to make in parallel. (Default: 3)

min_download_size = INT Minimum number of bytes we should ever try to download in a measurement. (Default: 1)

max_download_size = INT Maximum number of bytes we should ever try to download in a measurement. (Default: 1073741824) 1073741824 == 1 GiB

relayprioritizer

measure_authorities = {on, off} Whether or not to measure authorities. (Default: off)

fraction_relays = FLOAT The target fraction of best priority relays we would like to return. 0.05 is 5%. In a 7000 relay network, 5% is 350 relays. (Default: 0.05)

min_relays = INT The minimum number of best priority relays we are willing to return. (Default: 50)

cleanup

data_files_compress_after_days = INT After this many days, compress data files. (Default: 29)

data_files_delete_after_days = INT After this many days, delete data files. (Default: 57)

v3bw_files_compress_after_days = INT After this many days, compress v3bw files. (Default: 1)

v3bw_files_delete_after_days = INT After this many days, delete v3bw files. (Default: 7)

logging

to_file = {yes, no} Whether or not to log to a rotating file the directory paths.log_dname. (Default: yes)

to_stdout = {yes, no} Whether or not to log to stdout. (Default: yes)

to_syslog = {yes, no} Whether or not to log to syslog. (Default: no) NOTE that when sbws is launched by systemd, stdout goes to journal and syslog.

to_file_max_bytes = INT If logging to file, how large (in bytes) should the file be allowed to get before rotating to a new one. 10485760 is 10 MiB. If zero or number of backups is zero, never rotate the log file. (Default: 10485760)

to_file_num_backups = INT If logging to file, how many backups to keep. If zero or max bytes is zero, never rotate the log file. (Default: 50)

level = {debug, info, warning, error, critical} Level to log at. (Default: debug)

to_file_level = {debug, info, warning, error, critical} Level to log at when using files. (Default: debug)

to_stdout_level = {debug, info, warning, error, critical} Level to log at when using stdout. (Default: info)

to_syslog_level = {debug, info, warning, error, critical} Level to log at when using syslog. (Default: info)

format = STR Format string to use when logging. (Default: %(asctime)s %(module)s[%(process)s]: <%(levelname)s> %(message)s)

to_stdout_format = STR Format string to use when logging to stdout. (Default: ${format})

to_syslog_format = STR Format string to use when logging to syslog. (Default: %(module)s[%(process)s]: <%(levelname)s> %(message)s)

to_file_format = STR Format string to use when logging to files. (Default: %(asctime)s %(levelname)s %(threadName)s %(filename)s:%(lineno)s - %(funcName)s - %(message)s)

1.7.3 EXAMPLES

Example `destinations` section:

```
[scanner]
nickname = Manual
country = US

[destinations]
foo = on
bar = on
baz = off
```

(continues on next page)

(continued from previous page)

```
[destinations.foo]
# using HTTP
url = http://example.org/sbws.bin
country = ZZ
verify = False

[destinations.bar]
# using HTTPS
url = https://example.com/data
country = SN

[destinations.baz]
# this will be ignored
url = https://example.net/ask/stan/where/the/file/is.exe
country = TH
```

1.7.4 FILES

$HOME/.sbws.ini Default `sbws` user configuration path.

Any other path to the configuration file can be specified using the `sbws` argument `-c`

1.7.5 SEE ALSO

sbws (1), https://sbws.readthedocs.org.

1.7.6 BUGS

Please report bugs at https://gitlab.torproject.org/tpo/network-health/sbws/-/issues/.

CHAPTER 2

Developer/technical documentation

Included in the docs directory and in `sbws-doc` Debian package:

2.1 Contributing to Simple Bandwidth Scanner

Thank you for your interest in Simple Bandwidth Scanner (`sbws`).

Examples of contributions include:

- Bug reports, feature requests
- Code/documentation patches

2.1.1 Bug reports or feature requests

- Check that it has not been already reported.
- Open a issue in Tor Project Gitlab .

2.1.2 Code/documentation patches

The sbws canonical repository is https://gitweb.torproject.org/sbws.git, but we review patches using the Gitlab repository (https://gitlab.torproject.org/tpo/network-health/sbws/-/merge_requests) Merge Requests (MR).

To know more about `sbws` code,

See also:

- *Developer/technical documentation*
- `./docs/source/testing.rst` (or testing or *Installing tests dependencies and running tests*).
- `./docs/source/documenting.rst` (or documenting or *Installing and building the documentation*).

The following are guidelines we aim to follow.

Steps to create a MR

1. Create a issue in Tor Project Gitlab (*Open issue*)
2. Fork `sbws` via the Gitlab web interface: https://gitlab.torproject.org/tpo/network-health/sbws
3. Clone the repository locally
4. Install `sbws` as explained in ./INSTALL.rst and ./TESTING.rst Use `pip install -e <>`
5. If needed install the documentation and build it as explained in ./DOCUMENTATION.rst
6. Create a new feature branch. If the issue solves a bug, base the branch on the latest maintained version, eg. `maint-1.1` and name it with the name of the base branch plus `_bugXXX`, where *XXX* is the number of the issue. If the issue is a new feature, base the branch on the *master* branch and name it `ticketXXX`. Optionally, the last part of the branch name can be any string, eg. `maint-1.1_bugXXX_contributing`.
7. Write code (*Code style*), tests, documentation, extra files (*Extra required files*), commit (*Commits*), etc.
8. Ensure tests pass (./TESTING.rst).
9. Push your branch to your Gitlab repository.
10. Ensure the CI tests are passing (https://gitlab.torproject.org/tpo/network-health/sbws/-/pipelines)

Finally:

11. Create a MR from your branch at https://gitlab.torproject.org/tpo/network-health/sbws

Code style

Follow the Zen of Python (**PEP 20**)

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
```

Code should adhere to the **PEP 8** guidelines. Before release 1.0.0, some guidelines have not been followed, such as the ordering the inputs (**PEP 8#imports**).

External link: Code Style

All functions, methods and classes should have **PEP 0257** (except `__repr__` and `__str__`). Before release 1.0.0, some docstrigs do not have 3 double quotes `"""` (**PEP 0257#id15**).

External link: Documentation

New features should add a corresponding documentation in /docs.

An editor compatible with EditorConfig will help you to follow the general formatting code style.

Timestamps must be in UTC. It is prefered to use `datetime` objects or Unix timestamps. Timestamps read by the user should be always formatted in ISO 8601

Functional style is prefered:

- use list comprenhensions lambda, map, reduce

- avoid reasigigning variables, instead create new ones
- use `deepcopy` when passing list of objects to a function/method
- classes should change attributes only in one method (other than __init__?)

[FUNC]

In general, do not reinvent the wheel, use Python native modules as `logging`, instead of implementing similar functionality. Or use other packages when the new dependency can be extra, for instance vulture.

Extra required files

Any non-trivial change should contain tests. See ./TESTING.rst. When running tests, currently `flake8` informs on some PEP8 errors/warnings, but not all.

Commits

Each commit should reference the Tor Project Gitlab issue (example: `#12345`) and possibly the bugfix version. The commit message should contain `Closes: #bugnumber`.

From version 1.0.2 we started to prefix the summary with the subpackage or component, though we have not standarized the words to use, eg: `scanner`, `generate`, `v3bwfile`, `relaylist`, `doc`, `test`, `CI`.

From version 1.0.3, we also started to prefix the summary with `new`, `fix` or `chg`, so that gitchangelog automatically generates different sections in the CHANGELOG.

From version 1.1.0 we started to use the words `new`, `chg` and `fix`, not in the sense gitchangelog use them, but to match sematic versioning changes major, minor and patch.

Try to make each commit a logically separate changes.:

```
As a general rule, your messages should start with a single line that's
o more than about 50 characters and that describes the changeset concisely,
followed by a blank line, followed by a more detailed explanation.
The Git project requires that the more detailed explanation include
your motivation for the change and contrast its implementation with
previous behavior--this is a good guideline to follow.
It's also a good idea to use the imperative present tense in these messages.
In other words, use commands.
Instead of "I added tests for" or "Adding tests for," use "Add tests for."
```

[DIST]

Template originally written by Tim Pope: *example commit*

Code being reviewed workflow

When a MR is being reviewed, new changes might be needed:

- If the change does not modify a previous change, create new commits and push.
- If the change modifies a previous change and it's small, git commit fixup should be used. When it is agreed that the MR is ready, create a new branch named `mybranch_02` and run:

  ```
  rebase --autosquash
  ```

  push, create new MR and close old MR mentioning the number of the new MR.

- If the review takes long and when it's ready code related to the MR has changed in master, create a new branch named `mybranch_02` and run:

```
rebase master
```

  push, create new MR and close old MR mentioning the number of the new MR.

[MERG]

2.1.3 Reviewing code

All code should be peer-reviewed. Two reasons for this are:

```
Because a developer cannot think of everything at once;
Because a fresh pair of eyes may spot an error, a corner-case in the code,
insufficient documentation, a missing consistency check, etc.
```

[REVI]

Reviewers:

- Should let the contributor know what to improve/change.
- Should not push code to the contributor's branch.
- Should wait for contributor's changes or feedback after changes are requested, before merging or closing a MR.
- Should merge (not rebase) the MR.
- If rebase is needed due to changes in master, the contributor should create a new branch named *xxx_rebased* based on the reviewed branch, rebase and create a new MR from it, as explained above.
- If new changes are needed when the contributor's branch is ready to merge, the reviewer can create a new branch based on the contributor's branch, push the changes and merge that MR. The contributor should be notified about it.
- If the reviewer realize that new changes are needed after the MR has been merged, the reviewer can push to master, notifying the contributor about the changes.
- Because currently there are not many reviewers, reviewers can merge their own MR if there was not any feedback after a week.
- Should not push directly to master, unless changes are trivial (typos, extra spaces, etc.)
- Should not push to master new features while there are open MRs to review.

Currently, the reviewers are gk, ahf, juga.

2.1.4 Releases

Releases follow semantic versioning. Until release 1.0.0 is reached, this project is not considered production ready.

Currently development happens in master, this might change from release 1.0.0

so that master has the last release changes, and development happens in the next release branch.

Before major releases, ensure that:

- Installation from scratch, as specified in ./INSTALL.md, must success.
- All tests must pass.

- Tor must be able to parse the produced bw files (current way is manual)

  Todo: Test that run Tor as dirauth and parse the files

- Bandwidth files must produce graphs compatible with Torflow (current way to test it is manual)

  Todo: Implement something to compare error with current consensus.

- A dirauth should be able to understand the documentation, otherwise the documentation should be clarified.

Create a ./CHANGELOG.rst file. Each entry should reference the Tor Project Gitlab issue (example: `#12345`) and possibly the bugfix version. Until version 1.0.2 we have followed keep a changelog format.

From version 1.1.x, run `./scripts/maint/release.py` to create new releases. It uses gitchangelog to automatically add new CHANGELOG entries from the commits' messages.

2.1.5 Example commit message

```
Short (50 chars or less) summary of changes

More detailed explanatory text, if necessary.  Wrap it to
about 72 characters or so.  In some contexts, the first
line is treated as the subject of an email and the rest of
the text as the body.  The blank line separating the
summary from the body is critical (unless you omit the body
entirely); tools like rebase can get confused if you run
the two together.

Further paragraphs come after blank lines.

  - Bullet points are okay, too

  - Typically a hyphen or asterisk is used for the bullet,
    preceded by a single space, with blank lines in
    between, but conventions vary here
```

External eferences

2.2 Installing tests dependencies and running tests

To run the tests, extra Python depenencies are needed:

- Flake8
- tox
- pytest
- coverage

To install them from `sbws`

```
pip install .[dev] && pip install .[test]
```

To run the tests:

```
tox
```

2.3 Installing and building the documentation

To build the documentation, extra Python dependencies are needed:

- Sphinx
- recommonmark
- Pylint (only to update the diagrams)

To install them from `sbws`:

```
pip install .[doc]
```

To build the documentation as HTML:

```
cd docs/ && make html
```

The generated HTML will be in `docs/build/`.

To build the manual (`man`) pages:

```
cd docs/ && make man
```

The generated man pages will be in `docs/man/`.

To build the documentation diagrams:

```
cd docs/ && make umlsvg
```

The generated diagrams will be in `docs/build/_images/`.

To convert the `LaTeX` mathematical formulae to images, extra system dependencies are needed:

- Core and Extra Tex Live packages
- dvipng

They are included in most distributions. In Debian install them running:

```
apt install texlive-latex-extra dvipng
```

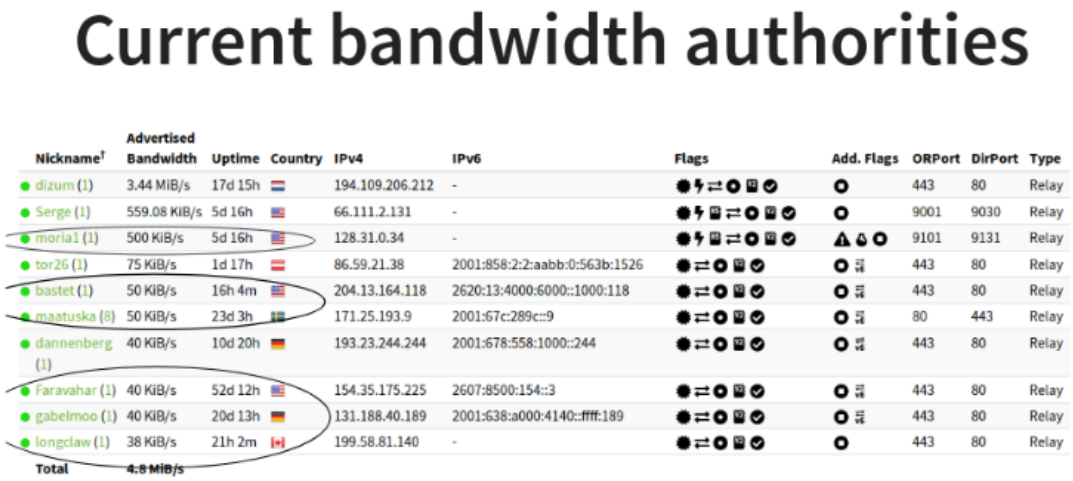
2.4 How sbws works

2.4.1 Overview

The *scanner* measures the bandwidth of each relay in the Tor network (except the directory authorities) by creating a two hops circuit with the relay. It then measures the bandwidth by downloading data from a *destination* Web Server and stores the measurements.

The *generator* read the measurements, aggregates, filters and scales them using torflow's scaling method.

Then it generates a *bandwidth list file* that is read by a *directory authority* to report relays' bandwidth in its vote.



2.4.2 Intialization

1. Parse the command line arguments and configuration files.
2. Launch a Tor thread with an specific configuration or connect to a running Tor daemon that is running with a suitable configuration.
3. Obtain the list of relays in the Tor network from the Tor consensus and descriptor documents.
4. Read and parse the old bandwidth measurements stored in the file system.
5. Select a subset of the relays to be measured next, ordered by:
    1. relays not measured.
    2. measurements age.

Classes used in the initialization:

Source code: *sbws.core.scanner.run_speedtest()*

2.4.3 Measuring relays

1. For every relay:
2. Select a second relay to build a Tor circuit.
3. Build the circuit.
4. Make HTTPS GET requests to the Web server over the circuit.

5. Store the time the request took and the amount of bytes requested.

Source code: *sbws.core.scanner.measure_relay()*

Measuring a relay

Source code: *sbws.core.scanner.measure_relay()*

2.4.4 Selecting a second relay

1. If the relay to measure is an exit, use it as an exit and obtain the non-exits.
2. If the relay to measure is not an exit, use it as first hop and obtain the exits.
3. From non-exits or exits, select one randomly from the ones that have double consensus bandwidth than the relay to measure.
4. If there are no relays that satisfy this, lower the required bandwidth.

Source code: *sbws.core.scanner.measure_relay()*

2.4.5 Selecting the data to download

1. While the downloaded data is smaller than 1GB or the number of download is minor than 5:
2. Randomly, select a 16MiB range.
3. If it takes less than 5 seconds, select a bigger range and don't keep any information.
4. If it takes more than 10 seconds, select an smaller range and don't keep any information.
5. Store the number of bytes downloaded and the time it took.

Source code: `sbws.core.scanner._should_keep_result()`

2.4.6 Writing the measurements to the filesystem

For every measured relay, the measurement result is put in a queue. There's an independent thread getting measurements from the queue every second. Every new measurement is appended to a file as a json line (but the file itself is not json!). The file is named with the current date. Every day a new file is created.

Source code: *sbws.lib.resultdump.ResultDump.enter()*

See also:

How aggregation and scaling works.

2.5 How aggregation and scaling works

See also:

How sbws works (scanner part).

Every hour, the generator:

1. Aggregate all the measurements (not older than 6 six days) for every relay.
2. Filter the measurements
3. Scale the measurements
4. Write the bandwidth file

Source code: *sbws.lib.v3bwfile.V3BWFile.from_results()*

2.5.1 Filtering the bandwidth measurements

Each relay bandwidth measurements are selected in the following way:

1. At least two bandwidth measurements (`Result` s) MUST have been obtained within an arbitrary number of seconds (currently one day). If they are not, the relay MUST NOT be included in the Bandwith File.
2. The measurements than are are older than an arbitrary number of senconds in the past MUST be discarded. Currently this number is the same as `data_period` (5 days) when not scaling as Torflow and 28 days when scaling as Torflow.

If the number of relays to include in the Bandwidth File are less than a percententage (currently 60%) than the number of relays in the consensus, additional Header Lines MUST be added (see XXX) to the Bandwith File and the relays SHOULD NOT be included.

2.5.2 Scaling the bandwidth measurements

Consensus bandwidth obtained by new implementations MUST be comparable to the consensus bandwidth, therefore they MUST implement torflow_scaling.

The bandwidth_file_spec appendix B describes torflow scaling and a linear scaling method.

See also:

Torflow aggregation and scaling and *Differences between Torflow and sbws*.

2.5.3 Writing the bandwidth file

The bandwidth file format is defined in the bandwidth_file_spec.

2.6 Torflow aggregation and scaling

See also:

How aggregation and scaling works and *Differences between Torflow and sbws*.

Torflow aggregation or scaling goal is:

From Torflow's README.spec.txt (section 2.2):

```
In this way, the resulting network status consensus bandwidth values
are effectively re-weighted proportional to how much faster the node
was as compared to the rest of the network.
```

2.6.1 Initialization

Constants in consensus that Torflow uses and don't change:

```
bandwidth-weights Wbd=0 Wbe=0 [] Wbm=10000 Wdb=10000 Web=10000 Wed=10000 Wee=10000␣
↪Weg=10000 Wem=10000 Wgb=10000 Wgd=0 Wgg=5852 [] Wmb=10000 Wmd=0 Wme=0 [] Wmm=10000

params [] bwauthpid=1
```

Constants in the code:

```
IGNORE_GUARD = 0
GUARD_SAMPLE_RATE = 2*7*24*60*60 # 2wks
MAX_AGE = 2*GUARD_SAMPLE_RATE;  # 4 weeks

K_p = 1.0
T_i = 0
T_i_decay = 0
T_d = 0
```

Initialization `ConsensusJunk`:

```
self.bwauth_pid_control = True
self.group_by_class = False
self.use_pid_tgt = False
self.use_circ_fails = False
self.use_best_ratio = True
self.use_desc_bw = True
self.use_mercy = False
self.guard_sample_rate = GUARD_SAMPLE_RATE
self.pid_max = 500.0
self.K_p = K_p = 1.0
self.T_i = T_i = 0
self.T_d = T_d = 0
self.T_i_decay = T_i_decay = 0

self.K_i = 0
self.K_d = self.K_p*self.T_d = 0
```

Initialization `Node`:

```
self.sbw_ratio = None
self.fbw_ratio = None
self.pid_bw = 0
self.pid_error = 0
self.prev_error = 0
self.pid_error_sum = 0
self.pid_delta = 0
self.ratio = None
self.new_bw = None
self.use_bw = -1
self.flags = ""

# measurement vars from bwauth lines
self.measured_at = 0
self.strm_bw = 0
self.filt_bw = 0
self.ns_bw = 0
self.desc_bw = 0
self.circ_fail_rate = 0
self.strm_fail_rate = 0
self.updated_at = 0
```

2.6.2 Descriptor values for each relay

From TorCtl.py code, it is the minimum of all the descriptor bandwidth values:

```
bws = map(int, g)
bw_observed = min(bws)

[snip]

return Router(ns.idhex, ns.nickname, bw_observed, dead, exitpolicy,
ns.flags, ip, version, os, uptime, published, contact, rate_limited,
ns.orhash, ns.bandwidth, extra_info_digest, ns.unmeasured)
```

`ns.bandwidth` is the consensus bandwidth, already multiplied by 1000:

```
yield NetworkStatus(*(m.groups()+(flags,)+(int(w.group(1))*1000,))+(unmeasured,))
```

Because of the matched regular expression, `bws` is **not** all the descriptor bandwidth values, but the average bandwidth and the observed bandwidth, ie., it does not take the average burst, what seems to be a bug in Torflow.

Eg. `bandwidth` line in a descriptor:

```
bandwidth 1536000 4096000 1728471
```

Only takes the first and last values, so:

```
bw_observed = min(bandwidth-avg, bandwidth-observed)
```

This is passed to `Router`, in which the descriptors bandwidth is assigned to the consensus bandwidth when there is no consensus bandwidth:

```
(idhex, name, bw, down, exitpolicy, flags, ip, version, os, uptime,
   published, contact, rate_limited, orhash,
   ns_bandwidth,extra_info_digest,unmeasured) = args

[snip]

if ns_bandwidth != None:
  self.bw = max(ns_bandwidth,1) # Avoid div by 0
else:
  self.bw = max(bw,1) # Avoid div by 0

[snip]

self.desc_bw = max(bw,1) # Avoid div by 0
```

So:

```
self.bw = ns_bwandwidth or min(bandwidth-avg, bandwidth-observed) or 1
desc_bw = min(bandwidth-avg, bandwidth-observed) or 1
```

And written by SQLSupport.py as descriptor and conensus bandwidth:

```
f.write(" desc_bw="+str(int(cvt(s.avg_desc_bw,0))))
f.write(" ns_bw="+str(int(cvt(s.avg_bw,0)))+"\n")
```

Descriptor bandwidth with PID control

Even though README.spec.txt talks about the consensus bandwidth, in aggregate.py code, the consensus bandwidth is never used, since `use_desc_bw` is initialized to True and never changed:

```
if cs_junk.bwauth_pid_control:
  if cs_junk.use_desc_bw:
    n.use_bw = n.desc_bw
  else:
    n.use_bw = n.ns_bw
```

So:

```
n.use_bw = n.desc_bw = min(bandwidth-avg, bandwidth-observed) or 1
```

2.6.3 Scaling the raw measurements

Overview

This diagram also includes *Descriptor bandwidth with PID control*, *Ratio for each relay* and *Scaled bandwidth for each relay with PID control*.

Simplified image from:

./_images/activity_torflow_scaling_simplified.svg

./_images/activity_torflow_scaling.svg

Stream and filtered bandwidth for each relay

They are calculated in the same way whether or not PID controller feedback is used.

From Torflow's README.spec.txt (section 1.6):

```
The strm_bw field is the average (mean) of all the streams for the relay
identified by the fingerprint field.

The filt_bw field is computed similarly, but only the streams equal to
or greater than the strm_bw are counted in order to filter very slow
streams due to slow node pairings.
```

In the code, SQLSupport.py, `strm_bw` is `sbw` and `filt_bw` is `filt_sbws`:

```
  for s in rs.router.streams:
    if isinstance(s, ClosedStream):
      tot_bytes += s.tot_bytes()
      tot_duration += s.end_time - s.start_time
      tot_bw += s.bandwidth()
      s_cnt += 1
  # FIXME: Hrmm.. do we want to do weighted avg or pure avg here?
  # If files are all the same size, it shouldn't matter..
  if s_cnt > 0:
    rs.sbw = tot_bw/s_cnt
  else: rs.sbw = None

for rs in RouterStats.query.filter(stats_clause).\
      options(eagerload_all('router.streams.circuit.routers')).all():
  tot_sbw = 0
  sbw_cnt = 0
  for s in rs.router.streams:
    if isinstance(s, ClosedStream):
      skip = False
      #for br in badrouters:
      #  if br != rs:
      #    if br.router in s.circuit.routers:
      #      skip = True
      if not skip:
        # Throw out outliers < mean
        # (too much variance for stddev to filter much)
        if rs.strm_closed == 1 or s.bandwidth() >= rs.sbw:
          tot_sbw += s.bandwidth()
          sbw_cnt += 1

if sbw_cnt: rs.filt_sbw = tot_sbw/sbw_cnt
else: rs.filt_sbw = None
```

When it is written to the file, it seem to write "None" string when `filt_sbw` or `strm_bw` are None. That would give an exception when calculating the network average. So it never happen?:

```
def cvt(a,b,c=1):
  if type(a) == float: return int(round(a/c,b))
  elif type(a) == int: return a
  elif type(a) == type(None): return "None"
```

(continues on next page)

(continued from previous page)

```
  else: return type(a)

f.write(" strm_bw="+str(cvt(s.sbw,0)))
f.write(" filt_bw="+str(cvt(s.filt_sbw,0)))
```

This is also expressed in pseudocode in the bandwidth file spec, section B.4 step 1.

Calling `bwstrm_i` to `strm_bw` and `bwfilt_i` to `filt_bw`, if `bw_j` is a measurement for a relay `i`, then::

```
bwstrm_i = mean(bw_j)  # for a relay, the average of all its measurements
bwfilt_i = mean(max(bwstrm_i, bw_j))
```

Stream and filtered bandwidth for all relays

From README.spec.txt (section 2.1):

```
Once we have determined the most recent measurements for each node, we
compute an average of the filt_bw fields over all nodes we have measured.
```

In Torflow's aggregate.py code:

```
for cl in ["Guard+Exit", "Guard", "Exit", "Middle"]:
  c_nodes = filter(lambda n: n.node_class() == cl, nodes.itervalues())
  if len(c_nodes) > 0:
    true_filt_avg[cl] = sum(map(lambda n: n.filt_bw, c_nodes))/float(len(c_nodes))
    true_strm_avg[cl] = sum(map(lambda n: n.strm_bw, c_nodes))/float(len(c_nodes))
    true_circ_avg[cl] = sum(map(lambda n: (1.0-n.circ_fail_rate),
                            c_nodes))/float(len(c_nodes))
```

The following code seems to be used only to log:

```
filt_avg = sum(map(lambda n: n.filt_bw, nodes.itervalues()))/float(len(nodes))
strm_avg = sum(map(lambda n: n.strm_bw, nodes.itervalues()))/float(len(nodes))
```

So it seems the `filt_avg` and `strm_avg` are calculated by class in both the cases with PID control and without PID control.

Calling `bwstrm` to `strm_avg` and `bwfilt` to `fitl_avg`, without taking into account the different types of nodes:

```
bwstrm = mean(bwstrm_i)
bwfilt = mean(bwfilt_i)
```

This is also expressed in pseudocode in the bandwidth file spec, section B.4 step 2.

Ratio for each relay

From README.spec.txt (section 2.2):

```
These averages are used to produce ratios for each node by dividing the
measured value for that node by the network average.
```

In Torflow's aggregate.py code:

```
for n in nodes.itervalues():
    n.fbw_ratio = n.filt_bw/true_filt_avg[n.node_class()]
    n.sbw_ratio = n.strm_bw/true_strm_avg[n.node_class()]

[snip]

# Choose the larger between sbw and fbw
  if n.sbw_ratio > n.fbw_ratio:
    n.ratio = n.sbw_ratio
  else:
    n.ratio = n.fbw_ratio
```

So:

```
n.ratio = max(sbw_ratio, n.fbw_ratio)
```

This is also expressed in pseudocode in the bandwidth file spec, section B.4 step 2 and 3.

Scaled bandwidth for each relay without PID control

From README.spec.txt (section 2.2):

```
These ratios are then multiplied by the most recent observed descriptor
bandwidth we have available for each node, to produce a new value for
the network status consensus process.
```

In aggregate.py code:

```
n.new_bw = n.desc_bw*n.ratio
```

So:

```
n.new_bw = (
    min(bandwidth-avg, bandwidth-observed) or 1 \
    * max(bwstrm_i / bwstrm, bwfilt_i / bwfilt_i)
)
```

This is also expressed in pseudocode in the bandwidth file spec, section B.4 step 5.

Scaled bandwidth for each relay with PID control

From README.spec.txt section 3.1:

```
The bandwidth authorities measure F_node: the filtered stream
capacity through a given node (filtering is described in Section 1.6).

[snip]

pid_error = e(t) = (F_node - F_avg)/F_avg.

[snip]

new_consensus_bw = old_consensus_bw +
                     old_consensus_bw * K_p * e(t) +
                     old_consensus_bw * K_i * \integral{e(t)} +
```

(continues on next page)

(continued from previous page)

```
                    old_consensus_bw * K_d * \derivative{e(t)}

[snip]

For the case where K_p = 1, K_i=0, and K_d=0, it can be seen that this
system is equivalent to the one defined in 2.2, except using consensus
bandwidth instead of descriptor bandwidth:

    new_bw = old_bw + old_bw*e(t)
    new_bw = old_bw + old_bw*(F_node/F_avg - 1)
    new_bw = old_bw*F_node/F_avg
    new_bw = old_bw*ratio
```

In Torflow's code, this is actually the case and most of the code is not executed because the default `K` values.

It seems then that `F_node` is `filt_bw` in Torflow's code or `bwfilt_i` here, and `F_avg` is `filt_avg` in Torflow's code or `bwfilt` here.

In aggregate.py code, pid error also depends on which of the ratios is greater:

```
if cs_junk.use_best_ratio and n.sbw_ratio > n.fbw_ratio:
        n.pid_error = (n.strm_bw - true_strm_avg[n.node_class()]) \
                      / true_strm_avg[n.node_class()]
        else:
        n.pid_error = (n.filt_bw - true_filt_avg[n.node_class()]) \
                      / true_filt_avg[n.node_class()]

[snip]

n.new_bw = n.use_bw + cs_junk.K_p*n.use_bw*n.pid_error
```

So:

```
if (bwstrm_i / bwstrm) > (bwfilt_i / bwfilt):
    pid_error = (bwstrm_i - bwstrm) / bwstrm = (bwstrm_i / bwstrm) - 1
else:
    pid_error = (bwfilt_i - bwfilt_i) / bwfilt = (bwfilt_i / bwfilt) - 1

new_bw = use_bw + use_bw * pid_error
```

Or:

```
if (bwstrm_i / bwstrm) > (bwfilt_i / bwfilt):
    new_bw = use_bw + use_bw * ((bwstrm_i / bwstrm) - 1)
    new_bw = use_bw + use_bw * (bwstrm_i / bwstrm) - use_bw
    new_bw = use_bw * (bwstrm_i / bwstrm)
else:
    new_bw = use_bw + use_bw * ((bwfilt_i / bwfilt) - 1)
    new_bw = use_bw + use_bw * (bwfilt_i / bwfilt) - use_bw
    new_bw = use_bw * (bwfilt_i / bwfilt)
```

Or:

```
new_bw = use_bw * max(bwstrm_i / bwstrm, bwfilt_i / bwfilt)
new_bw = (
    min(bandwidth-avg, bandwidth-observed) or 1
    * max(bwstrm_i / bwstrm, bwfilt_i / bwfilt)
)
```

Note: So, the new scaled bandwidth is the same for both cases with and without PID controller!

2.6.4 Other pid KeyValues in the Bandwidth File

Note: From the *Overview* it seems that the only variable needed to calculate the new scaled bandwidth is the `pid_error`, and from *Descriptor bandwidth with PID control*, it can be substituted by the stream and filtered bandwidths.

This are the variables that can then be ignored:

```
pid_error_sum
pid_delta
prev_error
```

2.6.5 Limit scaled bandwidth for each relay

It's calculated the same with and without PID control

Once each relay bandwidth is scaled, it is limited to a maximum, that is calculated as the sum of all the relays in the current consensus scaled bandwidth per 0.05.

From aggregate.py code:

```
NODE_CAP = 0.05

[snip]

if n.idhex in prev_consensus:
  if prev_consensus[n.idhex].bandwidth != None:
    prev_consensus[n.idhex].measured = True
    tot_net_bw += n.new_bw

[snip]

if n.new_bw > tot_net_bw*NODE_CAP:
  [snip]
  n.new_bw = int(tot_net_bw*NODE_CAP)
```

2.6.6 Round the scaled bandwidth for each relay

Finally, the new scaled bandwidth is expressed in kilobytes and rounded a number of digits.

2.7 Differences between Torflow and sbws

(Last updated 2020-02-18)

2.7.1 Aggregating measurements and scaling

Filtering

Torflow does not exclude relays because of having "few" measurements or "close" to each other for that relay, like sbws does *Filtering the bandwidth measurements*.

However this is currently disabled in sbws.

Network stream and filtered bandwidth

Torflow calculates the network stream and filtered averages by type of relay *Stream and filtered bandwidth for all relays*, while sbws is not taking into account the type of relay *Scaling the bandwidth measurements*.

Values from the previous Bandwidth File

sbws is not reading the previous Bandwidth File, but scaling all the values with the raw measurements.

Instead, Torflow uses the previous Bandwidth File values in some cases:

- When a relay measurement is older than the one in the previous Bandwidth File, it uses all the values from the previous Bandwidth File. (how is possible that the Bandwidth File would have a newer measurements?):

```
self.new_bw = prev_vote.bw * 1000
```

Bandwidth File KeyValues

sbws does not calculate nor write to the Bandwidth file the `pid` variables and KeyValues that are used in Torflow. Example of Torflow KeyValues not in sbws:

```
measured_at=1613547098 updated_at=1613547098 pid_error=11.275680184 pid_error_sum=11.
↪275680184 pid_bw=23255048 pid_delta=11.0140582849 circ_fail=0.0
```

sbws does not have `measured_at` and `updated_at` either.

Currently the scaled bandwidth in Torflow does not depend on those extra values and they seem to be just informative.

2.8 Code design

Todo:

- Link to refactor proposal.
- Change this page when refactoring is implemented.

2.8.1 UML classes diagram

classes_original.svg

2.8.2 Packages diagram

packages_sbws.svg

2.8.3 scanner threads

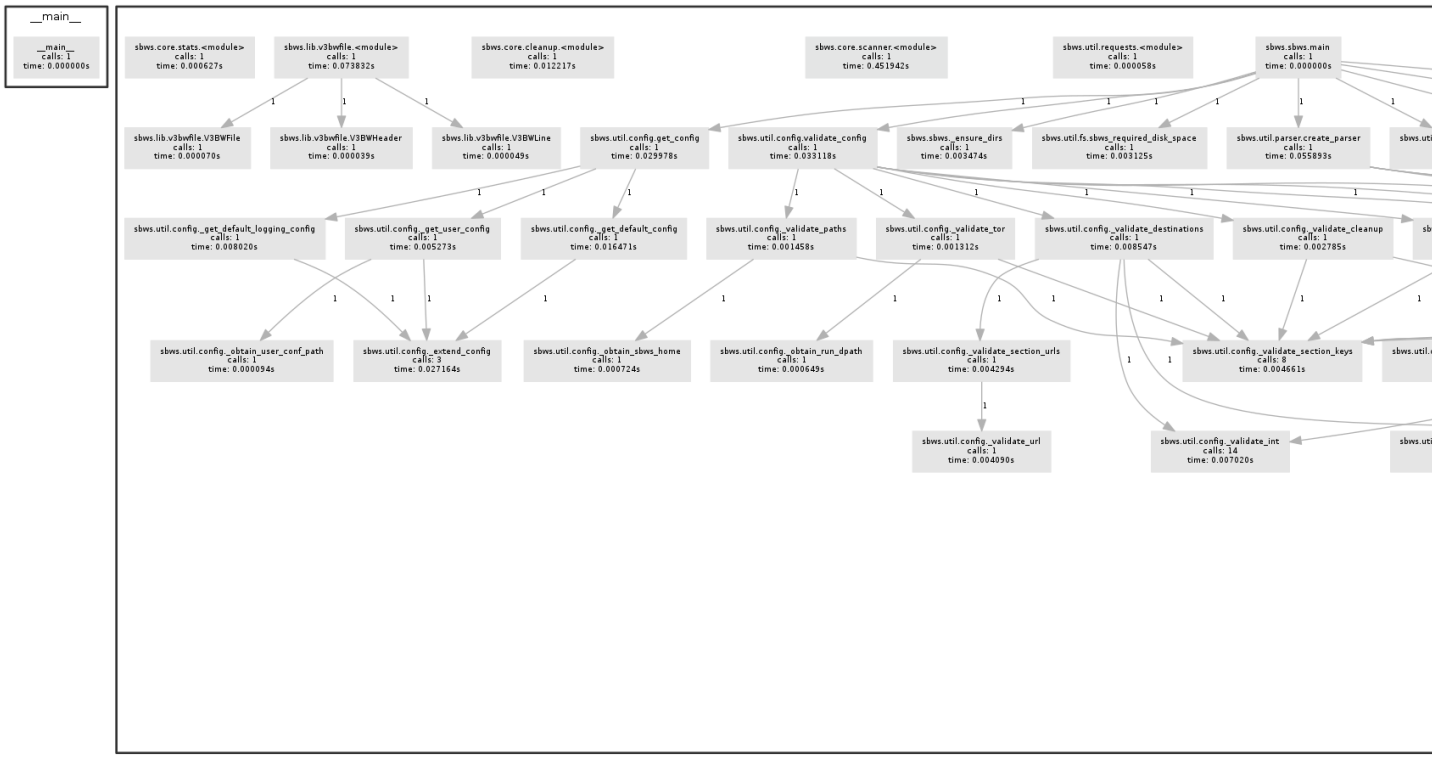
- *TorEventListener*: the thread that runs Tor and listens for events.
- ResultDump: the thread that get the measurement results from a queue every second.
- *multiprocessing.ThreadPool* starts 3 independent threads: - workers_thread - tasks_thread - results_thread
- measurement threads: they execute `sbws.core.scanner.measure_relay()` There'll be a maximum of 3 by default.

2.8.4 Critical sections

Data types that are read or wrote from the threads.

2.8.5 Call graph

Initialization calls to the moment where the measurement threads start.



callgraph.png

2.9 The `state.dat` file

This file contains state that multiple sbws commands may want access to and that needs to persist across processes. Both read and write access to this file is wrapped in the `State` class, allowing for safe concurrent access: the file is locked before reading or writing, and (for now) only simple data types are allowed so we can be sure to update the state file on disk every time the state is modified in memory.

At the time of writing, the following fields can exist in the state file.

2.9.1 `scanner_started`

The last time `sbws scanner` was started.

- **Producer**: `sbws scanner`, once at startup.
- **Consumer**: `sbws generate`, once each time it is ran.

Code: *`sbws.util.state.State`*

2.10 Internal code configuration files

Sbws has two default config files it reads: one general, and one specific to logging. They all get combined internally to the same `conf` structure.

It first reads the config file containing the default values for almost all options. If you installed sbws in a virtual environment located at /tmp/venv, then you will probably find the `config.default.ini` in a place such as `/tmp/venv/lib/python3.5/site-packages/sbws/` **You should never edit this file**. The contents of this default config file can be found *at the bottom of this page*.

Second, `sbws` will read `config.log.default.ini`. It will be located in the same place as the previous file, and **should not be edited** like the previous file. The contents of this default log config file can be found *at the bottom of this page*. Options set here overwrite options set in the previous config file.

Sbws then reads your custom config file. By default, it will search for it in `~/.sbws.ini`. Options in this file overwrite options set in previously read config files.

The user example config file provided by `sbws` might look like this.

Listing 2.1: Example sbws.example.ini

```
# Minimum configuration that needs to be customized
[scanner]
# A human-readable string with chars in a-zA-Z0-9 to identify your scanner
nickname = sbws_default
# ISO 3166-1 alpha-2 country code where the Web server destination is located.
# Default AA, to detect it was not edited.
country = SN

[destinations]
# With several destinations, the scanner can continue even if some of them
# fail, which can be caused by a network problem on their side.
# If all of them fail, the scanner will stop, which
```

(continues on next page)

(continued from previous page)

```
# will happen if there is network problem on the scanner side.

# A destination can be disabled changing `on` by `off`
foo = on

[destinations.foo]
# the domain and path to the 1GB file.
url = https://example.com/does/not/exist.bin
# Whether to verify or not the TLS certificate. Default True
verify = False
# ISO 3166-1 alpha-2 country code where the Web server destination is located.
# Default AA, to detect it was not edited.
# Use ZZ if the location is unknown (for instance, a CDN).
country = ZZ

# Number of consecutive times that a destination could not be used to measure
# before stopping to try to use it for a while that by default is 3h.
max_num_failures = 3

## The following logging options are set by default.
## There is no need to change them unless other options are prefered.
; [logging]
; # Whether or not to log to a rotating file the directory paths.log_dname
; to_file = yes
; # Whether or not to log to stdout
; to_stdout = yes
; # Whether or not to log to syslog
; # NOTE that when sbws is launched by systemd, stdout goes to journal and
; # syslog.
; to_syslog = no

; # Level to log at. Debug, info, warning, error, critical.
; # `level` must be set to the lower of all the handler levels.
; level = debug
; to_file_level = debug
; to_stdout_level = info
; to_syslog_level = info
; # Format string to use when logging
; format = %(module)s[%(process)s]: <%(levelname)s> %(message)s
; # verbose formatter useful for debugging
; to_file_format = %(asctime)s %(levelname)s %(threadName)s %(filename)s:%(lineno)s -
↪%(funcName)s - %(message)s
; # Not adding %(asctime)s to to stdout since it'll go to syslog when using
; # systemd, and it'll have already the date.
; to_stdout_format = ${format}
; to_syslog_format = ${format}

# To disable certificate validation, uncomment the following
# verify = False
```

No other configuration files are read.

2.10.1 Default Configuration

Listing 2.2: config.default.ini

```
[paths]
sbws_home = ~/.sbws
datadir = ${sbws_home}/datadir
v3bw_dname = ${sbws_home}/v3bw
# The latest bandwidth file is atomically symlinked to
# V3BandwidthsFile ${v3bw_dname}/latest.v3bw
v3bw_fname = ${v3bw_dname}/{}.v3bw
state_fname = ${sbws_home}/state.dat
log_dname = ${sbws_home}/log

[destinations]
# How often to check if a destional is usable
usability_test_interval = 300

[general]
# Days into the past that measurements are considered valid
data_period = 5
# Timeout in seconds to give to the python Requests library. It MUST be a
# single float. Requests will use it both as the connect() timeout and the
# timeout between bytes received from the server. See
# http://docs.python-requests.org/en/master/user/advanced/#timeouts
http_timeout = 10
# Timeout in seconds for waiting on a circuit to be built. It MUST be an
# **int**. We will use this both as the CircuitBuildTimeout and a timeout
# to give to stem for waiting on a circuit to be built since
# CircuitBuildTimeout doesn't handle the case of a TLS connection to a relay
# taking forever, and probably other not-yet-discovered cases.
circuit_timeout = 60
# Whether or not to reset the bandwidth measurements when the relay's IP
# address changes. If it changes, we only consider results for the relay that
# we obtained while the relay was located at its most recent IP address.
# This is NOT implemented for IPv6.
reset_bw_ipv4_changes = off
reset_bw_ipv6_changes = off

[scanner]
# A human-readable string with chars in a-zA-Z0-9 to identify your scanner
nickname = IDidntEditTheSBWSConfig
# ISO 3166-1 alpha-2 country code. To be edited.
# Default to a non existing country to detect it was not edited.
country = AA
# Limits on what download times are too fast/slow/etc.
download_toofast = 1
download_min = 5
download_target = 6
download_max = 10
# How many RTT measurements to make
num_rtts = 0
# Number of downloads with acceptable times we must have for a relay before
# moving on
num_downloads = 5
# The number of bytes to initially request from the server
initial_read_request = 16384
# How many measurements to make in parallel
measurement_threads = 3
```

(continues on next page)

(continued from previous page)

```
# Minimum number of bytes we should ever try to download in a measurement
min_download_size = 1
# Maximum number of bytes we should ever try to download in a measurement
# 1073741824 == 1 GiB
max_download_size = 1073741824

[tor]
datadir = ${paths:sbws_home}/tor
run_dpath = ${datadir}
control_socket = ${tor:run_dpath}/control
pid = ${tor:run_dpath}/tor.pid
# note this is a directory
log = ${tor:datadir}/log
external_control_port =
extra_lines =

[cleanup]
# After this many days, compress data files
# #40017: To generate files as Torflow the result files must be kept for
# GENERATE_PERIOD seconds.
# The number of days after they are compressed or deleted could be added
# as defaults (currently globals.py), and just as a factor of GENERATE_PERIOD.
data_files_compress_after_days = 29
# After this many days, delete data files.
# 57 == 28 * 2 + 1.
data_files_delete_after_days = 57
# After this many days, compress v3bw files (1d)
v3bw_files_compress_after_days = 1
# After this many days, delete v3bw files (7d)
v3bw_files_delete_after_days = 7

[relayprioritizer]
# Whether or not to measure authorities
measure_authorities = off
# The target fraction of best priority relays we would like to return.
# 0.05 is 5%. In a 7000 relay network, 5% is 350 relays.
#
# In a network of ~6500 relays and with a ResultDump containing 1 result per
# relay, the best_priority() function takes ~11 seconds to complete on my
# home desktop. Using this parameter allows us to balance between calling
# best_priority() more often (but wasting more CPU), and calling it less
# often (but taking longer to get back to relays with non-successful results).
#
# Alternatively, we could rewrite best_priority() to not suck so much.
fraction_relays = 0.05
# The minimum number of best priority relays we are willing to return
min_relays = 50

[logging]
# Whether or not to log to a rotating file the directory paths.log_dname
to_file = yes
# Whether or not to log to stdout
to_stdout = yes
# Whether or not to log to syslog
# NOTE that when sbws is launched by systemd, stdout goes to journal and
# syslog.
to_syslog = no
```

(continues on next page)

(continued from previous page)

```
# If logging to file, how large (in bytes) should the file be allowed to get
# before rotating to a new one. 10485760 is 10 MiB. If zero or number of
# backups is zero, never rotate the log file.
to_file_max_bytes = 10485760
# If logging to file, how many backups to keep. If zero or max bytes is zero,
# never rotate the log file.
to_file_num_backups = 50
# Level to log at. Debug, info, warning, error, critical.
# `level` must be set to the lower of all the handler levels.
level = info
to_file_level = info
to_stdout_level = info
to_syslog_level = info
# Format string to use when logging
format = %(asctime)s %(module)s[%(process)s]: <%(levelname)s> (%(threadName)s)
↪%(message)s
to_stdout_format = ${format}
to_syslog_format = %(module)s[%(process)s]: <%(levelname)s> %(message)s
# verbose formatter useful for debugging
to_file_format = %(asctime)s %(levelname)s (%(threadName)s) %(filename)s:%(lineno)s -
↪%(funcName)s - %(message)s
```

If you know how to use Python's logging configuration file format, then you can override or add to what is listed here by editing your config file.

Listing 2.3: config.log.default.ini

```
[loggers]
keys = root,sbws

[handlers]
keys = to_file,to_stdout,to_syslog

[formatters]
keys = to_file,to_stdout,to_syslog

[logger_root]
level = WARNING
handlers = to_file
propagate = 1
qualname=root

[logger_sbws]
propagate = 0
qualname=sbws

[handler_to_stdout]
class = StreamHandler
formatter = to_stdout
args = (sys.stdout,)

[handler_to_file]
class = handlers.RotatingFileHandler
formatter = to_file
args = ('/dev/null', )
```

(continues on next page)

(continued from previous page)

```
# for logging to system log
[handler_to_syslog]
class=handlers.SysLogHandler
formatter=to_syslog
args = ('/dev/log',)

[formatter_to_stdout]
# format date as syslog and journal
datefmt = %b %d %H:%M:%S

[formatter_to_file]
datefmt = %b %d %H:%M:%S

[formatter_to_syslog]
```

2.11 Internal Tor configuration for the scanner

The scanner needs a specific Tor configuration. The following options are either set when launching Tor or required when connection to an existing Tor daemon.

Default configuration:

- `SocksPort auto`: To proxy requests over Tor.
- `CookieAuthentication 1`: The easiest way to authenticate to Tor.
- `UseEntryGuards 0`: To avoid path bias warnings.
- `UseMicrodescriptors 0`: Because full server descriptors are needed.
- `SafeLogging 0`: Useful for logging, since there's no need for anonymity.
- `LogTimeGranularity 1`
- `ProtocolWarnings 1`
- `FetchDirInfoEarly 1`
- `FetchDirInfoExtraEarly 1`: Respond to *MaxAdvertisedBandwidth* as soon as possible.
- `FetchUselessDescriptors 1`: Keep fetching descriptors, even when idle.
- `LearnCircuitBuildTimeout 0`: To keep circuit build timeouts static.

Configuration that depends on the user configuration file:

- `CircuitBuildTimeout ...`: The timeout trying to build a circuit.
- `DataDirectory ...`: The Tor data directory path.
- `PidFile ...`: The Tor PID file path.
- `ControlSocket ...`: The Tor control socket path.
- `Log notice ...`: The Tor log level and path.

Configuration that needs to be set on runtime:

- `__DisablePredictedCircuits 1`: To build custom circuits.
- `__LeaveStreamsUnattached 1`: The scanner is attaching the streams itself.

Configuration that can be set on runtime and fail:

- `ConnectionPadding 0`: Useful for avoiding extra traffic, since scanner anonymity is not a goal.

Currently most of the code that sets this configuration is in *sbws.util.stem.launch_tor()* and the default configuration is `sbws/globals.py`.

Note: the location of this code is being refactored.

2.12 Package API

2.12.1 Subpackages

sbws.core package

Submodules

sbws.core.cleanup module

Util functions to cleanup disk space.

`sbws.core.cleanup.`**`gen_parser`**(*sub*)
Helper function for the broader argument parser generating code that adds in all the possible command line arguments for the cleanup command.

Parameters **sub** (*argparse._SubParsersAction*) – what to add a sub-parser to

`sbws.core.cleanup.`**`main`**(*args*, *conf*)
Main entry point in to the cleanup command.

Parameters

- **args** (*argparse.Namespace*) – command line arguments
- **conf** (*configparser.ConfigParser*) – parsed config files

sbws.core.generate module

`sbws.core.generate.`**`gen_parser`**(*sub*)

`sbws.core.generate.`**`main`**(*args*, *conf*)

sbws.core.scanner module

Measure the relays.

`sbws.core.scanner.`**`create_path_relay`**(*relay*, *dest*, *rl*, *cb*, *relay_as_entry=True*)

`sbws.core.scanner.`**`dispatch_worker_thread`**(**a*, ***kw*)

`sbws.core.scanner.`**`dumpstacks`**()

`sbws.core.scanner.`**`error_no_circuit`**(*circ_fps*, *nicknames*, *reason*, *relay*, *dest*, *our_nick*)

`sbws.core.scanner.`**`error_no_helper`**(*relay*, *dest*, *our_nick=''*)

sbws.core.scanner.**force_get_results**(*pending_results*)
Try to get either the result or an exception, which gets logged.

It is call by *wait_for_results()* when the time waiting for the results was long.

To get either the *Result* or an exception, call `get()` with timeout. Timeout is low since we already waited.

`get` is not call before, because it blocks and the callbacks are not call.

sbws.core.scanner.**gen_parser**(*sub*)

sbws.core.scanner.**get_random_range_string**(*content_length*, *size*)
Return a random range of bytes of length **size**. **content_length** is the size of the file we will be requesting a range of bytes from.

For example, for content_length of 100 and size 10, this function will return one of the following: '0-9', '1-10', '2-11', [...] '89-98', '90-99'

sbws.core.scanner.**main**(*args*, *conf*)

sbws.core.scanner.**main_loop**(*args*, *conf*, *controller*, *relay_list*, *circuit_builder*, *result_dump*, *relay_prioritizer*, *destinations*, *pool*)
Starts and reuse the threads that measure the relays forever.

It starts a loop that will be run while there is not and event signaling that sbws is stopping (because of SIGTERM or SIGINT).

Then, it starts a second loop with an ordered list (generator) of relays to measure that might a subset of all the current relays in the Network.

For every relay, it starts a new thread which runs `measure_relay` to measure the relay until there are `max_pending_results` threads. After that, it will reuse a thread that has finished for every relay to measure. It is the the pool method `apply_async` which starts or reuse a thread. This method returns an `ApplyResult` immediately, which has a `ready` methods that tells whether the thread has finished or not.

When the thread finish, ie. `ApplyResult` is `ready`, it triggers `result_putter` callback, which put the `Result` in `ResultDump` queue and complete immediately.

`ResultDump` thread (started before and out of this function) will get the `Result` from the queue and write it to disk, so this doesn't block the measurement threads.

If there was an exception not catched by `measure_relay`, it will call instead `result_putter_error`, which logs the error and complete immediately.

Before the outer loop iterates, it waits (non blocking) that all the `Results` are ready calling `wait_for_results`. This avoid to start measuring the same relay which might still being measured.

sbws.core.scanner.**measure_bandwidth_to_server**(*session*, *conf*, *dest*, *content_length*)

Returns tuple results or None if the if the measurement fail. None or exception if the measurement fail.

sbws.core.scanner.**measure_relay**(*args*, *conf*, *destinations*, *cb*, *rl*, *relay*)
Select a Web server, a relay to build the circuit, build the circuit and measure the bandwidth of the given relay.

Return Result a measurement Result object

sbws.core.scanner.**measure_rtt_to_server**(*session*, *conf*, *dest*, *content_length*)
Make multiple end-to-end RTT measurements by making small HTTP requests over a circuit + stream that should already exist, persist, and not need rebuilding. If something goes wrong and not all of the RTT measurements can be made, return None. Otherwise return a list of the RTTs (in seconds).

Returns tuple results or None if the if the measurement fail. None or exception if the measurement fail.

sbws.core.scanner.**result_putter**(*result_dump*)
Create a function that takes a single argument – the measurement result – and return that function so it can be used by someone else

sbws.core.scanner.**result_putter_error**(*target*)
Create a function that takes a single argument – an error from a measurement – and return that function so it can be used by someone else

sbws.core.scanner.**run_speedtest**(*args*, *conf*)
Initializes all the data and threads needed to measure the relays.

It launches or connect to Tor in a thread. It initializes the list of relays seen in the Tor network. It starts a thread to read the previous measurements and wait for new measurements to write them to the disk. It initializes a class that will be used to order the relays depending on their measurements age. It initializes the list of destinations that will be used for the measurements. It initializes the thread pool that will launch the measurement threads. The pool starts 3 other threads that are not the measurement (worker) threads. Finally, it calls the function that will manage the measurement threads.

sbws.core.scanner.**stop_threads**(*signal*, *frame*, *exit_code=0*)

sbws.core.scanner.**timed_recv_from_server**(*session*, *dest*, *byte_range*)
Request the **byte_range** from the URL at **dest**. If successful, return True and the time it took to download. Otherwise return False and an exception.

sbws.core.scanner.**wait_for_results**(*num_relays_to_measure*, *pending_results*)
Wait for the pool to finish and log progress.

While there are relays being measured, just log the progress and sleep `TIMEOUT_MEASUREMENTS` (3mins), which is aproximately the time it can take to measure a relay in the worst case.

When there has not been any relay measured in `TIMEOUT_MEASUREMENTS` and there are still relays pending to be measured, it means there is no progress and call *force_get_results()*.

This can happen in the case of a bug that makes either *measure_relay()*, *result_putter()* (callback) and/or *result_putter_error()* (callback error) stall.

Note: in a future refactor, this could be simpler by:

1. Initializing the pool at the begingging of each loop
2. Callling `close()`; `join()` after `apply_async()`, to ensure no new jobs are added until the pool has finished with all the ones in the queue.

As currently, there would be still two cases when the pool could stall:

1. There's an exception in `measure_relay` and another in `callback_err`
2. There's an exception `callback`.

This could also be simpler by not having callback and callback error in `apply_async` and instead just calling callback with the `pending_results`.

(callback could be also simpler by not having a thread and queue and just storing to disk, since the time to write to disk is way smaller than the time to request over the network.)

---

sbws.core.stats module

sbws.core.stats.**gen_parser**(*sub*)
Helper function for the broader argument parser generating code that adds in all the possible command line

arguments for the stats command.

Parameters **sub** (*argparse._SubParsersAction*) – what to add a sub-parser to

sbws.core.stats.**main**(*args*, *conf*)
Main entry point into the stats command.

Parameters

- **args** (*argparse.Namespace*) – command line arguments
- **conf** (*configparser.ConfigParser*) – parsed config files

sbws.core.stats.**print_stats**(*args*, *data*)
Called from main to print various statistics about the organized **data** to stdout.

Parameters

- **args** (*argparse.Namespace*) – command line arguments
- **data** (*dict*) – keyed by relay fingerprint, and with values of *sbws.lib.resultdump.Result* subclasses

Module contents

sbws.lib package

Submodules

sbws.lib.circuitbuilder module

class sbws.lib.circuitbuilder.**CircuitBuilder**(*args*, *conf*, *controller*, *relay_list=None*, *close_circuits_on_exit=True*)
Bases: object

The CircuitBuilder interface.

Subclasses must implement their own build_circuit() function. Subclasses may keep additional state if they'd find it helpful.

The primary way to use a CircuitBuilder of any type is to simply create it and then call cb.build_circuit(...) with any options that your CircuitBuilder type needs.

It might be good practice to close circuits as you find you no longer need them, but CircuitBuilder will keep track of existing circuits and close them when it is deleted.

close_circuit(*circ_id*)

class sbws.lib.circuitbuilder.**GapsCircuitBuilder**(**a*, ***kw*)
Bases: *sbws.lib.circuitbuilder.CircuitBuilder*

Same as CircuitBuilder but implements build_circuit.

build_circuit(*path*)
Return parent class build circuit method.

Since sbws is only building 2 hop paths, there is no need to add random relays to the path, or convert back and forth between fingerprint and Relay objects.

sbws.lib.circuitbuilder.**valid_circuit_length**(*path*)

sbws.lib.relaylist module

class sbws.lib.relaylist.**Relay**(*fp*, *cont*, *ns=None*, *desc=None*, *timestamp=None*)
Bases: object

address

average_bandwidth

burst_bandwidth

can_exit_to_port(*port*, *strict=False*)
Returns True if the relay has an exit policy and the policy accepts exiting to the given port or False otherwise.

If strict is true, it only returns the exits that can exit to all IPs and that port.

The exits that are IPv6 only or IPv4 but rejecting some public networks will return false. On July 2020, there were 67 out of 1095 exits like this.

If strict is false, it returns any exit that can exit to some public IPs and that port.

Note that the EXIT flag exists when the relay can exit to 443 **and** 80. Currently all Web servers are using 443, so it would not be needed to check the EXIT flag too, using this function.

consensus_bandwidth
Return the consensus bandwidth in Bytes.

Consensus bandwidth is the only bandwidth value that is in kilobytes.

consensus_bandwidth_is_unmeasured

consensus_valid_after
Obtain the consensus Valid-After from the document of this relay network status.

exit_policy

fingerprint

flags

increment_relay_recent_measurement_attempt()
Increment The number of times that a relay has been queued to be measured.

It is call from main_loop().

increment_relay_recent_priority_list()
The number of times that a relay is "prioritized" to be measured.

It is call from *best_priority()*.

is_exit_not_bad_allowing_port(*port*, *strict=False*)

last_consensus_timestamp

master_key_ed25519
Obtain ed25519 master key of the relay in server descriptors.

Returns str, the ed25519 master key base 64 encoded without trailing '='s.

nickname

observed_bandwidth

relay_in_recent_consensus_count
Number of times the relay was in a conensus.

relay_recent_measurement_attempt_count

relay_recent_priority_list_count

update_relay_in_recent_consensus(*timestamp=None*)

update_router_status(*router_status*)
Update this relay router status (from the consensus).

update_server_descriptor(*server_descriptor*)
Update this relay server descriptor (from the consensus.

class sbws.lib.relaylist.**RelayList**(*args*, *conf*, *controller*, *measurements_period=432000*, *state=None*)
Bases: object

Keeps a list of all relays in the current Tor network and updates it transparently in the background. Provides useful interfaces for getting only relays of a certain type.

authorities

bad_exits

exit_min_bw()

exits

exits_not_bad_allowing_port(*port*, *strict=False*)

fast

guards

increment_recent_measurement_attempt()
Increment the number of times that any relay has been queued to be measured.

It is call from main_loop().

It is read and stored in a state file.

last_consensus_timestamp
Returns the datetime when the last consensus was obtained.

non_exit_min_bw()

non_exits

random_relay()

recent_consensus_count
Number of times a new consensus was obtained.

recent_measurement_attempt_count

relays

relays_fingerprints

sbws.lib.relaylist.**valid_after_from_network_statuses**(*network_statuses*)
Obtain the consensus Valid-After datetime from the document attribute of a stem.descriptor.RouterStatusEntryV3.

Parameters **network_statuses** (*list*) –

returns datetime:

sbws.lib.relayprioritizer module

class sbws.lib.relayprioritizer.**RelayPrioritizer**(*args*, *conf*, *relay_list*, *result_dump*)
Bases: object

best_priority(*prioritize_result_error=False*, *return_fraction=True*)
Yields a new ordered list of relays to be measured next.

The relays that were measured farther away in the past, get prioritized (lowest priority number, first in the list). The relays that were measured more recently get lower priority (last in the list, higher priority number).

Optionally, the relays which measurements failed can be prioritized (first in the list). However, unstable relays that fail often to be measured, might fail again and stable relays will get measured only when their measurements become old enough. The opposite might be more suitable: give lower priority to the relays that are unstable, to don't spend time measuring relays that might fail to be measured.

Optionally, return only a fraction of all the relays in the network. Since there could be new relays in the network while measuring the list of relays returned by this method, this method is run again before all the relays in the network are measured.

Note: In a future refactor, instead of having a static fraction of relays to be measured, this method could be call when it're known that there're X number of new relays in the network.

Since measurements made before than X days ago (too old) are not considered, and the initial list of past measurements is only filtered when the scanner starts, it's needed to filter here again to discard those measurements.

Parameters

- **prioritize_result_error** (*bool*) – whether prioritize or not measurements that did not succed.
- **return_fraction** (*bool*) – whether to return only a fraction of the relays seen in the network or return all.

return: a generator of the new ordered list of relays to measure next.

increment_recent_priority_list()
Increment the number of times that *best_priority()* has been run.

increment_recent_priority_relay(*relays_count*)
Increment the number of relays that have been "prioritized" to be measured in a *best_priority()*.

recent_priority_list_count

recent_priority_relay_count

sbws.lib.resultdump module

class sbws.lib.resultdump.**Result**(*relay*, *circ*, *dest_url*, *scanner_nick*, *t=None*)
Bases: object

A bandwidth measurement for a relay.

It re-implements *Relay* as a inner class.

class Relay(*fingerprint*, *nickname*, *address*, *master_key_ed25519*, *average_bandwidth=None*, *burst_bandwidth=None*, *observed_bandwidth=None*, *consensus_bandwidth=None*, *consensus_bandwidth_is_unmeasured=None*, *relay_in_recent_consensus=None*, *relay_recent_measurement_attempt=None*, *relay_recent_priority_list=None*)

Bases: `object`

A Tor relay.

It re-implements *Relay* with the attributes needed.

Note: in a future refactor it would be simpler if a `Relay` has measurements and a measurement has a relay, instead of every measurement re-implementing `Relay`.

---

address

circ

consensus_bandwidth

consensus_bandwidth_is_unmeasured

dest_url

fingerprint

static from_dict(*d*)

Returns a *Result* subclass from a dictionary.

Returns None if the `version` attribute is not `RESULT_VERSION`

It raises `NotImplementedError` when the dictionary `type` can not be parsed.

Note: in a future refactor, the conversions to/from object-dictionary will be simpler using `setattr` and `__dict__`

`version` is not being used and should be removed.

---

master_key_ed25519

nickname

relay_average_bandwidth

relay_burst_bandwidth

relay_in_recent_consensus

Number of times the relay was in a consensus.

relay_observed_bandwidth

relay_recent_measurement_attempt

Returns the relay recent measurements attemps.

It is initialized in *Relay* and incremented in *main_loop()*.

relay_recent_priority_list

Returns the relay recent "prioritization"s to be measured.

It is initialized in *Relay* and incremented in *main_loop()*.

scanner

time

to_dict()

type

version

class sbws.lib.resultdump.**ResultDump**(*args*, *conf*)
Bases: object

Runs the enter() method in a new thread and collects new Results on its queue. Writes them to daily result files in the data directory

enter()
Main loop for the ResultDump thread.

When there are results in the queue, queue.get will get them until there are not anymore or timeout happen.

For every result it gets, it process it and store in the filesystem, which takes ~1 millisecond and will not trigger the timeout. It can then store in the filesystem ~1000 results per second.

I does not accept any other data type than Results or list of Results, therefore is not possible to put big data types in the queue.

If there are not any results in the queue, it waits 1 second and checks again.

handle_result(*result*)
Call from ResultDump thread. If we are shutting down, ignores ResultError* types

results_for_relay(*relay*)

store_result(*result*)
Call from ResultDump thread

class sbws.lib.resultdump.**ResultError**(**a*, *msg=None*, ***kw*)
Bases: *sbws.lib.resultdump.Result*

freshness_reduction_factor
When the RelayPrioritizer encounters this Result, how much should it adjust its freshness? (See RelayPrioritizer.best_priority() for more information about "freshness")

A higher factor makes the freshness lower (making the Result seem older). A lower freshness leads to the relay having better priority, and better priority means it will be measured again sooner.

The value 0.5 was chosen somewhat arbitrarily, but a few weeks of live network testing verifies that sbws is still able to perform useful measurements in a reasonable amount of time.

static from_dict(*d*)
Returns a *Result* subclass from a dictionary.

Returns None if the `version` attribute is not `RESULT_VERSION`

It raises `NotImplementedError` when the dictionary `type` can not be parsed.

Note: in a future refactor, the conversions to/from object-dictionary will be simpler using `setattr` and `__dict__`

`version` is not being used and should be removed.

---

msg

to_dict()

type

class sbws.lib.resultdump.**ResultErrorAuth**(*a, **kw)
Bases: *sbws.lib.resultdump.ResultError*

freshness_reduction_factor
Override the default ResultError.freshness_reduction_factor because a ResultErrorAuth is most likely not the measured relay's fault, so we shouldn't hurt its priority as much. A higher reduction factor means a Result's effective freshness is reduced more, which makes the relay's priority better.

The value 0.9 was chosen somewhat arbitrarily.

static from_dict(*d*)
Returns a *Result* subclass from a dictionary.

Returns None if the `version` attribute is not `RESULT_VERSION`

It raises `NotImplementedError` when the dictionary `type` can not be parsed.

Note: in a future refactor, the conversions to/from object-dictionary will be simpler using `setattr` and `__dict__`

`version` is not being used and should be removed.

---

to_dict()

type

class sbws.lib.resultdump.**ResultErrorCircuit**(*a, **kw)
Bases: *sbws.lib.resultdump.ResultError*

freshness_reduction_factor
There are a few instances when it isn't the relay's fault that the circuit failed to get built. Maybe someday we'll try detecting whose fault it most likely was and subclassing ResultErrorCircuit. But for now we don't. So reduce the freshness slightly more than ResultError does by default so priority isn't hurt quite as much.

A (hopefully very very rare) example of when a circuit would fail to get built is when the sbws client machine suddenly loses Internet access.

static from_dict(*d*)
Returns a *Result* subclass from a dictionary.

Returns None if the `version` attribute is not `RESULT_VERSION`

It raises `NotImplementedError` when the dictionary `type` can not be parsed.

Note: in a future refactor, the conversions to/from object-dictionary will be simpler using `setattr` and `__dict__`

`version` is not being used and should be removed.

---

to_dict()

type

class sbws.lib.resultdump.**ResultErrorDestination**(*a, **kw)
Bases: *sbws.lib.resultdump.ResultError*

Error when there is not a working destination Web Server.

It is instanciated in *measure_relay()*.

Note: this duplicates code and add more tech-debt, since it's the same as the other *ResultError* classes except for the `type`. In a future refactor, there should be only one `ResultError` class and assign the type in the `scanner` module.

static from_dict(*d*)
Returns a *Result* subclass from a dictionary.

Returns None if the `version` attribute is not `RESULT_VERSION`

It raises `NotImplementedError` when the dictionary `type` can not be parsed.

Note: in a future refactor, the conversions to/from object-dictionary will be simpler using `setattr` and `__dict__`

`version` is not being used and should be removed.

to_dict()

type

class sbws.lib.resultdump.**ResultErrorSecondRelay**(**a*, ***kw*)
Bases: *sbws.lib.resultdump.ResultError*

Error when it could not be found a second relay suitable to measure a relay.

A second suitable relay is a relay that: - Has at least equal bandwidth as the relay to measure. - If the relay to measure is not an exit, the second relay is an exit without *bad* flag and can exit to port 443. - If the relay to measure is an exit, the second relay is not an exit.

It is instanciated in *measure_relay()*.

Note: this duplicates code and add more tech-debt, since it's the same as the other *ResultError* classes except for the `type`. In a future refactor, there should be only one `ResultError` class and assign the type in the `scanner` module.

static from_dict(*d*)
Returns a *Result* subclass from a dictionary.

Returns None if the `version` attribute is not `RESULT_VERSION`

It raises `NotImplementedError` when the dictionary `type` can not be parsed.

Note: in a future refactor, the conversions to/from object-dictionary will be simpler using `setattr` and `__dict__`

`version` is not being used and should be removed.

to_dict()

type

class sbws.lib.resultdump.**ResultErrorStream**(**a*, ***kw*)
Bases: *sbws.lib.resultdump.ResultError*

static from_dict(*d*)
Returns a *Result* subclass from a dictionary.

Returns None if the `version` attribute is not `RESULT_VERSION`

It raises `NotImplementedError` when the dictionary `type` can not be parsed.

Note: in a future refactor, the conversions to/from object-dictionary will be simpler using `setattr` and `__dict__`

`version` is not being used and should be removed.

---

to_dict()

type

class sbws.lib.resultdump.**ResultSuccess**(*rtts*, *downloads*, **a*, ***kw*)

Bases: *sbws.lib.resultdump.Result*

downloads

static from_dict(*d*)

Returns a *Result* subclass from a dictionary.

Returns None if the `version` attribute is not `RESULT_VERSION`

It raises `NotImplementedError` when the dictionary `type` can not be parsed.

Note: in a future refactor, the conversions to/from object-dictionary will be simpler using `setattr` and `__dict__`

`version` is not being used and should be removed.

---

rtts

to_dict()

type

sbws.lib.resultdump.**load_recent_results_in_datadir**(*fresh_days*, *datadir*, *success_only=False*, *on_changed_ipv4=False*, *on_changed_ipv6=False*)

Given a data directory, read all results files in it that could have results in them that are still valid. Trim them, and return the valid Results as a list

sbws.lib.resultdump.**load_result_file**(*fname*, *success_only=False*)

Reads in all lines from the given file, and parses them into Result structures (or subclasses of Result). Optionally only keeps ResultSuccess. Returns all kept Results as a result dictionary. This function does not care about the age of the results

sbws.lib.resultdump.**merge_result_dicts**(*d1*, *d2*)

Given two dictionaries that contain Result data, merge them. Result dictionaries have keys of relay fingerprints and values of lists of results for those relays.

sbws.lib.resultdump.**trim_results**(*fresh_days*, *result_dict*)

Given a result dictionary, remove all Results that are no longer valid and return the new dictionary

sbws.lib.resultdump.**trim_results_ip_changed**(*result_dict*, *on_changed_ipv4=False*, *on_changed_ipv6=False*)

When there are results for the same relay with different IPs, create a new results' dictionary without that relay's results using an older IP.

Parameters

- **result_dict** (*dict*) – a dictionary of results
- **on_changed_ipv4** (*bool*) – whether to trim the results when a relay's IPv4 changes
- **on_changed_ipv6** (*bool*) – whether to trim the results when a relay's IPv6 changes

Returns a new results dictionary

sbws.lib.resultdump.**write_result_to_datadir**(*result*, *datadir*)
Can be called from any thread

sbws.lib.v3bwfile module

Classes and functions that create the bandwidth measurements document (v3bw) used by bandwidth authorities.

class sbws.lib.v3bwfile.**V3BWFile**(*v3bwheader*, *v3bwlines*)
Bases: object

Create a Bandwidth List file following spec version 1.X.X

Parameters

- **v3bwheader** (V3BWHeader) – header
- **v3bwlines** (*list*) – V3BWLines

static bw_kb(*bw_lines*, *reverse=False*)

bw_line_for_node_id(*node_id*)
Returns the bandwidth line for a given node fingerprint.

Used to combine data when plotting.

static bw_sbws_scale(*bw_lines*, *scale_constant=7500*, *reverse=False*)
Return a new V3BwLine list scaled using sbws method.

Parameters

- **bw_lines** (*list*) – bw lines to scale, not self.bw_lines, since this method will be before self.bw_lines have been initialized.
- **scale_constant** (*int*) – the constant to multiply by the ratio and the bandwidth to obtain the new bandwidth

Returns list V3BwLine list

static bw_torflow_scale(*bw_lines*, *desc_bw_obs_type=1*, *cap=0.05*, *num_round_dig=2*, *reverse=False*, *router_statuses_d=None*)
Obtain final bandwidth measurements applying Torflow's scaling method.

See details in *Torflow aggregation and scaling*.

classmethod from_results(*results*, *scanner_country=None*, *destinations_countries=None*, *state_fpath=''*, *scale_constant=7500*, *scaling_method=1*, *torflow_obs=0*, *torflow_cap=0.05*, *round_digs=2*, *secs_recent=None*, *secs_away=None*, *min_num=0*, *consensus_path=None*, *max_bw_diff_perc=50*, *reverse=False*)
Create V3BWFile class from sbws Results.

Parameters

- **results** (*dict*) – see below
- **state_fpath** (*str*) – path to the state file

- **scaling_method** (*int*) – Scaling method to obtain the bandwidth Possible values: {None, SBWS_SCALING, TORFLOW_SCALING} = {0, 1, 2}
- **scale_constant** (*int*) – sbws scaling constant
- **torflow_obs** (*int*) – method to choose descriptor observed bandwidth
- **reverse** (*bool*) – whether to sort the bw lines descending or not

Results are in the form:

```
{'relay_fp1': [Result1, Result2, ...],
 'relay_fp2': [Result1, Result2, ...]}
```

classmethod from_v100_fpath(*fpath*)

classmethod from_v1_fpath(*fpath*)

info_stats

static is_max_bw_diff_perc_reached(*bw_lines*, *max_bw_diff_perc=50*, *router_statuses_d=None*)

is_min_perc

max_bw

mean_bw

static measured_progress_stats(*num_bw_lines*, *number_consensus_relays*, *min_perc_reached_before*)

Statistics about measurements progress, to be included in the header.

Parameters

- **bw_lines** (*list*) – the bw_lines after scaling and applying filters.
- **consensus_path** (*str*) – the path to the cached consensus file.
- **state_fpath** (*str*) – the path to the state file

Returns dict, bool Statistics about the progress made with measurements and whether the percentage of measured relays has been reached.

median_bw

min_bw

num

static read_number_consensus_relays(*consensus_path*)

Read the number of relays in the Network from the cached consensus file.

static read_router_statuses(*consensus_path*)

Read the router statuses from the cached consensus file.

static set_under_min_report(*bw_lines*)

Mondify the Bandwidth Lines adding the KeyValue *under_min_report*, *vote*.

sum_bw

to_plt(*attrs=['bw'], sorted_by=None*)

Return bandwidth data in a format useful for matplotlib.

Used from external tool to plot.

update_progress(*num_bw_lines*, *header*, *number_consensus_relays*, *state*)
Returns True if the minimim percent of Bandwidth Lines was reached and False otherwise. Update the header with the progress.

static warn_if_not_accurate_enough(*bw_lines*, *scale_constant=7500*)

write(*output*)

class sbws.lib.v3bwfile.**V3BWHeader**(*timestamp*, ***kwargs*)
Bases: object

Create a bandwidth measurements (V3bw) header following bandwidth measurements document spec version 1.X.X.

Parameters

- **timestamp** (*str*) – timestamp in Unix Epoch seconds of the most recent generator result.
- **version** (*str*) – the spec version
- **software** (*str*) – the name of the software that generates this
- **software_version** (*str*) – the version of the software
- **kwargs** (*dict*) – extra headers. Currently supported:
  - earliest_bandwidth: str, ISO 8601 timestamp in UTC time zone when the first bandwidth was obtained
  - generator_started: str, ISO 8601 timestamp in UTC time zone when the generator started

add_relays_excluded_counters(*exclusion_dict*)
Add the monitoring KeyValues to the header about the number of relays not included because they were not eligible.

add_stats(***kwargs*)

add_time_report_half_network()
Add to the header the time it took to measure half of the network.

It is not the time the scanner actually takes on measuring all the network, but the number_eligible_relays that are reported in the bandwidth file and directory authorities will vote on.

This is calculated for half of the network, so that failed or not reported relays do not affect too much.

For instance, if there are 6500 relays in the network, half of the network would be 3250. And if there were 4000 eligible relays measured in an interval of 3 days, the time to measure half of the network would be 3 days * 3250 / 4000.

Since the elapsed time is calculated from the earliest and the latest measurement and a relay might have more than 2 measurements, this would give an estimate on how long it would take to measure the network including all the valid measurements.

Log also an estimated on how long it would take with the current number of relays included in the bandwidth file.

static consensus_count_from_file(*state_fpath*)

static earliest_bandwidth_from_results(*results*)

classmethod from_lines_v1(*lines*)

Parameters **lines** (*list*) – list of lines to parse

Returns tuple of V3BWHeader object and non-header lines

classmethod from_lines_v100(*lines*)

Parameters **lines** (*list*) – list of lines to parse

Returns tuple of V3BWHeader object and non-header lines

classmethod from_results(*results*, *scanner_country=None*, *destinations_countries=None*, *state_fpath=''*)

classmethod from_text_v1(*text*)

Parameters **text** (*str*) – text to parse

Returns tuple of V3BWHeader object and non-header lines

static generator_started_from_file(*state_fpath*)
ISO formatted timestamp for the time when the scanner process most recently started.

keyvalue_tuple_ls
Return list of all KeyValue tuples

keyvalue_unordered_tuple_ls
Return list of KeyValue tuples that do not have specific order.

keyvalue_v1str_ls
Return KeyValue list of strings following spec v1.X.X.

keyvalue_v2_ls
Return KeyValue list of strings following spec v2.X.X.

static latest_bandwidth_from_results(*results*)

num_lines

static recent_measurement_attempt_count_from_file(*state_fpath*)
Returns the number of times any relay was queued to be measured in the recent (by default 5) days from the state file.

static recent_priority_list_count_from_file(*state_fpath*)
Returns the number of times *best_priority()* was run in the recent (by default 5) days from the state file.

static recent_priority_relay_count_from_file(*state_fpath*)
Returns the number of times any relay was "prioritized" to be measured in the recent (by default 5) days from the state file.

strv1
Return header string following spec v1.X.X.

strv2
Return header string following spec v2.X.X.

class sbws.lib.v3bwfile.**V3BWLine**(*node_id*, *bw*, ***kwargs*)
Bases: object

Create a Bandwidth List line following the spec version 1.X.X.

Parameters

- **node_id** (*str*) – the relay fingerprint
- **bw** (*int*) – the bandwidth value that directory authorities will include in their votes.
- **kwargs** (*dict*) – extra headers.

Note: tech-debt: move node_id and bw to kwargs and just ensure that the required values are in `**kwargs`

bw_keyvalue_tuple_ls
Return list of KeyValue Bandwidth Line tuples.

bw_keyvalue_v1str_ls
Return list of KeyValue Bandwidth Line strings following spec v1.X.X.

static bw_mean_from_results(*results*)

static bw_median_from_results(*results*)

bw_strv1
Return Bandwidth Line string following spec v1.X.X.

static consensus_bandwidth_from_results(*results*)
Obtain the last consensus bandwidth from the results.

static consensus_bandwidth_is_unmeasured_from_results(*results*)
Obtain the last consensus unmeasured flag from the results.

del_relay_type()

static desc_bw_avg_from_results(*results*)
Obtain the last descriptor bandwidth average from the results.

static desc_bw_bur_from_results(*results*)
Obtain the last descriptor bandwidth burst from the results.

static desc_bw_obs_last_from_results(*results*)

static desc_bw_obs_mean_from_results(*results*)

classmethod from_bw_line_v1(*line*)

classmethod from_data(*data*, *fingerprint*)

classmethod from_results(*results*, *secs_recent=None*, *secs_away=None*, *min_num=0*, *router_statuses_d=None*)
Convert sbws results to relays' Bandwidth Lines

`bs` stands for Bytes/seconds `bw_mean` means the bw is obtained from the mean of the all the downloads' bandwidth. Downloads' bandwidth are calculated as the amount of data received divided by the the time it took to received. bw = data (Bytes) / time (seconds)

static last_time_from_results(*results*)

static result_types_from_results(*results*)

static results_away_each_other(*results*, *secs_away=None*)

static results_recent_than(*results*, *secs_recent=None*)

static rtt_from_results(*results*)

set_relay_type(*relay_type*)

sbws.lib.v3bwfile.**kb_round_x_sig_dig**(*bw_bs*, *digits=2*)
Convert bw_bs from bytes to kilobytes, and round the result to 'digits' significant digits. Results less than or equal to 1 are rounded up to 1. Returns an integer.

digits must be greater than 0. n must be less than or equal to 2**82, to avoid floating point errors.

sbws.lib.v3bwfile.**num_results_of_type**(*results*, *type_str*)

sbws.lib.v3bwfile.**result_type_to_key**(*type_str*)

sbws.lib.v3bwfile.**round_sig_dig**(*n*, *digits=2*)
Round n to 'digits' significant digits in front of the decimal point. Results less than or equal to 1 are rounded to 1. Returns an integer.

digits must be greater than 0. n must be less than or equal to 2**73, to avoid floating point errors.

Module contents

sbws.util package

Submodules

sbws.util.config module

Util functions to manage sbws configuration files.

sbws.util.config.**configure_logging**(*args*, *conf*)

sbws.util.config.**get_config**(*args*)
Get ConfigParser interpolating all configuration files.

sbws.util.config.**validate_config**(*conf*)
Checks the given conf for bad values or bad combinations of values. If there's something wrong, returns False and a list of error messages. Otherwise, return True and an empty list

sbws.util.filelock module

class sbws.util.filelock.**DirectoryLock**(*dname*)
Bases: sbws.util.filelock._FLock

Holds a lock on a file in **dname** so that other sbws processes/threads won't try to read/write while we are reading/writing in this directory.

```
>>> with DirectoryLock(dname):
>>>     # do some reading/writing in dname
>>> # no longer have the lock
```

Note: The directory must already exist.

Parameters **dname** (*str*) – Name of directory for which we want to obtain a lock

class sbws.util.filelock.**FileLock**(*fname*)
Bases: sbws.util.filelock._FLock

Holds a lock on **fname** so that other sbws processes/threads won't try to read/write while we are reading/writing this file.

```
>>> with FileLock(fname):
>>>     # do some reading/writing of fname
>>> # no longer have the lock
```

Parameters **fname** (*str*) – Name of the file for which we want to obtain a lock

sbws.util.parser module

sbws.util.parser.**create_parser**()

sbws.util.state module

class sbws.util.state.**State**(*fname*)

Bases: object

json wrapper to read a json file every time it gets a key and to write to the file every time a key is set.

Every time a key is got or set, the file is locked, to atomically access and update the file across threads and across processes.

```
>>> state = State('foo.state')
>>> # state == {}
```

```
>>> state['linux'] = True
>>> # 'foo.state' now exists on disk with the JSON for {'linux': True}
```

```
>>> # We read 'foo.state' from disk in order to get the most up-to-date
>>> #     state info. Pretend another process has updated 'linux' to be
>>> #     False
>>> state['linux']
>>> # returns False
```

```
>>> # Pretend another process has added the user's age to the state file.
>>> #     As before, we read the state file from disk for the most
>>> #     up-to-date info.
>>> state['age']
>>> # Returns 14
```

```
>>> # We now set their name. We read the state file first, set the option,
>>> #     and then write it out.
>>> state['name'] = 'John'
```

```
>>> # We can do many of the same things with a State object as with a dict
>>> for key in state: print(key)
>>> # Prints 'linux', 'age', and 'name'
```

count(*k*)

Returns the length if the key value is a list or the sum of number if the key value is a list of list or the key value or None if the state doesn't have the key.

get(*key*, *d=None*)

Implements a dictionary `get` method reading and locking a json file.

sbws.util.stem module

sbws.util.stem.**add_event_listener**(*controller*, *func*, *event*)

sbws.util.stem.**attach_stream_to_circuit_listener**(*controller*, *circ_id*)

Returns a function that should be given to add_event_listener(). It looks for newly created streams and attaches them to the given circ_id

sbws.util.stem.**circuit_str**(*controller*, *circ_id*)

sbws.util.stem.**get_socks_info**(*controller*)
Returns the first SocksPort Tor is configured to listen on, in the form of an (address, port) tuple

sbws.util.stem.**init_controller**(*conf*)

sbws.util.stem.**is_bootstrapped**(*c*)

sbws.util.stem.**is_torrc_starting_point_set**(*tor_controller*)
Verify that the tor controller has the correct configuration.

When connecting to a tor controller that has not been launched by sbws, it should have been configured to work with sbws.

sbws.util.stem.**launch_or_connect_to_tor**(*conf*)

sbws.util.stem.**launch_tor**(*conf*)

sbws.util.stem.**only_relays_with_bandwidth**(*controller*, *relays*, *min_bw=None*, *max_bw=None*)
Given a list of relays, only return those that optionally have above **min_bw** and optionally have below **max_bw**, inclusively. If neither min_bw nor max_bw are given, essentially just returns the input list of relays.

sbws.util.stem.**parse_user_torrc_config**(*torrc*, *torrc_text*)
Parse the user configuration torrc text call *extra_lines* to a dictionary suitable to use with stem and return a new torrc dictionary that merges that dictionary with the existing torrc. Example:

```
[tor]
extra_lines =
    Log debug file /tmp/tor-debug.log
    NumCPUs 1
```

sbws.util.stem.**remove_event_listener**(*controller*, *func*)

sbws.util.stem.**rs_relay_type**(*rs*)

sbws.util.stem.**set_torrc_options_can_fail**(*controller*)
Set options that can fail, at runtime.

They can be set at launch, but since the may fail because they are not supported in some Tor versions, it's easier to try one by one at runtime and ignore the ones that fail.

sbws.util.stem.**set_torrc_runtime_options**(*controller*)
Set torrc options at runtime.

sbws.util.stem.**set_torrc_starting_point**(*controller*)
Set the torrc starting point options.

sbws.util.userquery module

sbws.util.userquery.**query_yes_no**(*question*, *default='yes'*)
Ask a yes/no question via input() and return the user's answer.

Parameters

- **question** (*str*) – Prompt given to the user.
- **default** (*str*) – The assumed answer if th user just hits **Enter**. It must be `'yes'` (the default if no default is given), `'no'`, or `None` (meaning an answer is required from the user).

Returns `True` if we ended up with a 'yes' answer, otherwise `False`.

Module contents

2.12.2 Submodules

2.12.3 sbws.globals module

sbws.globals.**fail_hard**(**a*, ***kw*)
Log something . . . and then exit as fast as possible

sbws.globals.**touch_file**(*fname*, *times=None*)
If **fname** exists, update its last access and modified times to now. If **fname** does not exist, create it. If **times** are specified, pass them to os.utime for use.

Parameters

- **fname** (*str*) – Name of file to update or create
- **times** (*tuple*) – 2-tuple of floats for access time and modified time respectively

2.13 Implementation decissions

2.13.1 Dependencies

When a needed feature is already implemented in some other software, there're usually some things to consider whether to use that software as dependency or re-implement the feature:

Possible advantages using other software:

- zero maintainance
- not reinventing the wheel

Possible disadvantages using other software:

- maybe too big
- maybe introduce security issues
- maybe is not maintaned

sbws version

Because some bwauths install sbws from the git repository, it is useful to know from which git revision they install it from. We'd prefer to do not see the git revision when it is installed from a git tag or Debian package (which is usually built from a git tag or git archive release).

A first solution would be to obtain the git revision at runtime, but:

- sbws is not usually running from the same directory as the git repository, as the installation might install it in other directory.
- if some other git repository is the current path, sbws might be obtaining the git revision of that other repository.

So next solution was to obtain the git revision at build/install time. To achive this, an script should be called from the installer or at runtime whenever *__version__* needs to be read.

While it could be implemented by us, there're two external tools that achive this.

setuptools_scm

https://github.com/pypa/setuptools_scm/

Advantages:

- does what we want, for 19 commits after 1.1.0 tag it'd add an string like *'1.1.1.dev19+g76ef2fe0.d20200221*. We don't need the date, but it can probably be removed and it does not hurt.
- we don't need to maintain it.

Disadvantages:

- it adds the extra dependency setuptools_scm.
- it does not obtain the version from a git archive, though there's other tool that does that.
- the version reported comes only from build time, so if we make a commit without running *setup.py*, sbws will not report the new version.

versioneer

https://github.com/warner/python-versioneer

Advantages:

- it does not add any extra dependency. The first time, versioneer needs to be installed. When run, it will generate *versioneer.py* and *_version.py*, which are created from versioneer itself. Then it can be uninstall
- does what we want, for 19 commits after 1.1.0 tag it'd add an string like *1.1.0+19.g76ef2fe0*. Note the difference with *1.1.0* from he *1.1.1* generated by
- we don't need to maintain it.
- it is also capable to obtain the version from a git archive.
- the version reported at build time and runtime is the same.

Disadvantages:

- it adds extra code to sbws and it's quite a lot
- the generated code is independent from the upstream and loses the tests.
- does not seem maintained.

Conclussion

Because *setuptools_scm* gives only the version at build time, we decided to use *versioneer*. We might need to change it in the future if starts giving problems with other git or python versions or we find a way to make *setuptools_scm* to detect the same version at buildtime and runtime.

See https://github.com/MartinThoma/MartinThoma.github.io/blob/1235fcdecda4d71b42fc07bfe7db327a27e7bcde/content/2018-11-13-python-package-versions.md for other comparative versioning python packages.

2.13.2 Changing Bandwidth file monitoring KeyValues

In version 1.1.0 we added KeyValues call `recent_X_count` and `relay_X_count` which implied to modify serveral parts of the code.

We only stored numbers for simpliciy, but then the value of this numbers accumulate over the time and there is no way to know to which number decrease since some of the main objects are not recreated at runtime and do not have attributes about when they were created or updated. The relations between the object do no follow usual one-to-many or many-to-many relationships either, to be able to induce some numbers from the related objects.

The only way we could think to solve this is to store list of timestamps, instead of just numbers, as an attribute in the objects that need to store some counting.

Where the values of the keys come from?

In the file system, there are only two types of files were these values can be stored: - the results files in `datadir` - the `state.dat` file

Because of the structure of the content in the results files, they can store KeyValues for the relays, but not for the headers, which need to be stored in the `state.dat` file.

The classes that manage these KeyValues are:

`RelayList`:

- recent_consensus_count
- recent_measurement_attempt_count

`RelayPrioritizer`:

- recent_priority_list_count
- recent_priority_relay_count

`Relay` and `Result`:

- relay_in_recent_consensus_count
- relay_recent_measurement_attempt_count
- relay_recent_priority_list_count

Transition from numbers to datetimes

The KeyValues named `_count` in the results and the state will be ignored when sbws is restarted with this change, since they will be written without `_count` names in these files json .

We could add code to count this in the transition to this version, but these numbers are wrong anyway and we don't think it's worth the effort since they will be correct after 5 days and they have been wrong for long time.

Additionally `recent_measurement_failure_count` will be negative, since it's calculated as `recent_measurement_attempt_count` minus all the results. While the total number of results in the last 5 days is corrrect, the number of the attempts won't be until 5 days have pass.

Disadvantages

`sbws generate`, with 27795 measurement attempts takes 1min instead of a few seconds. The same happens with the `RelayPrioritizer.best_priority`, though so far that seems ok since it's a python generator in a thread and the measurements start before it has calculated all the priorities. The same happens with the `ResultDump` that read/write the data in a thread.

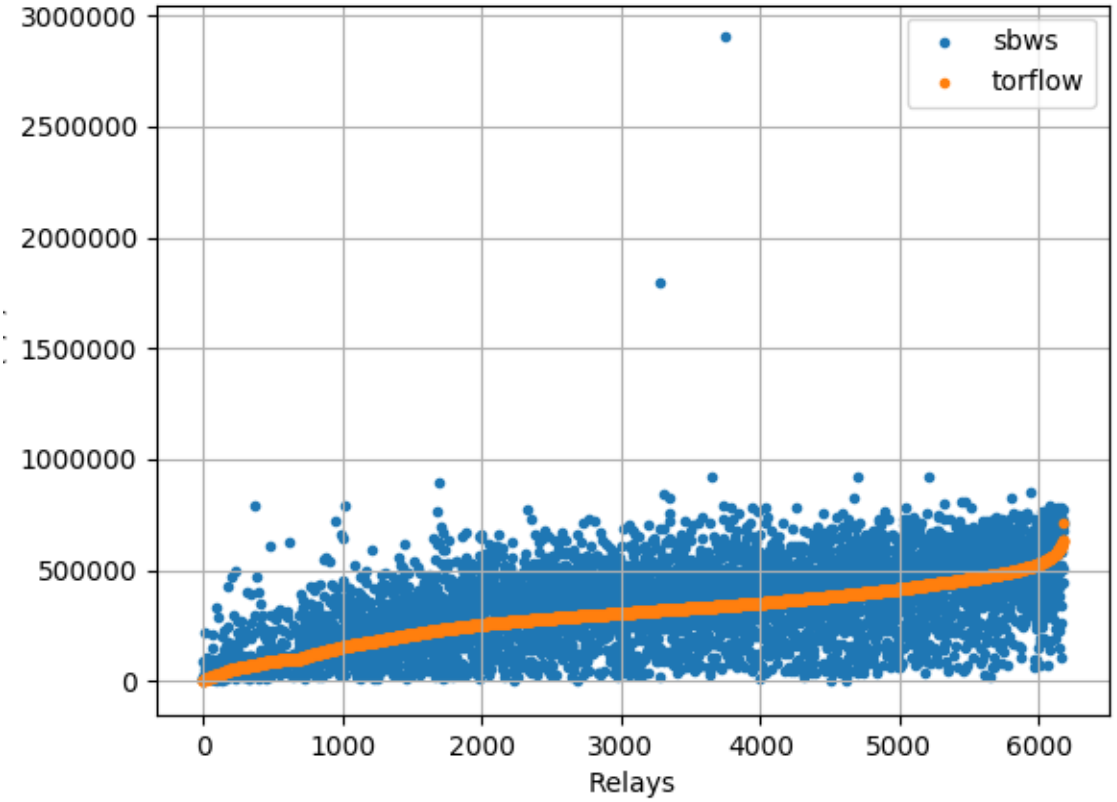
Conclusion

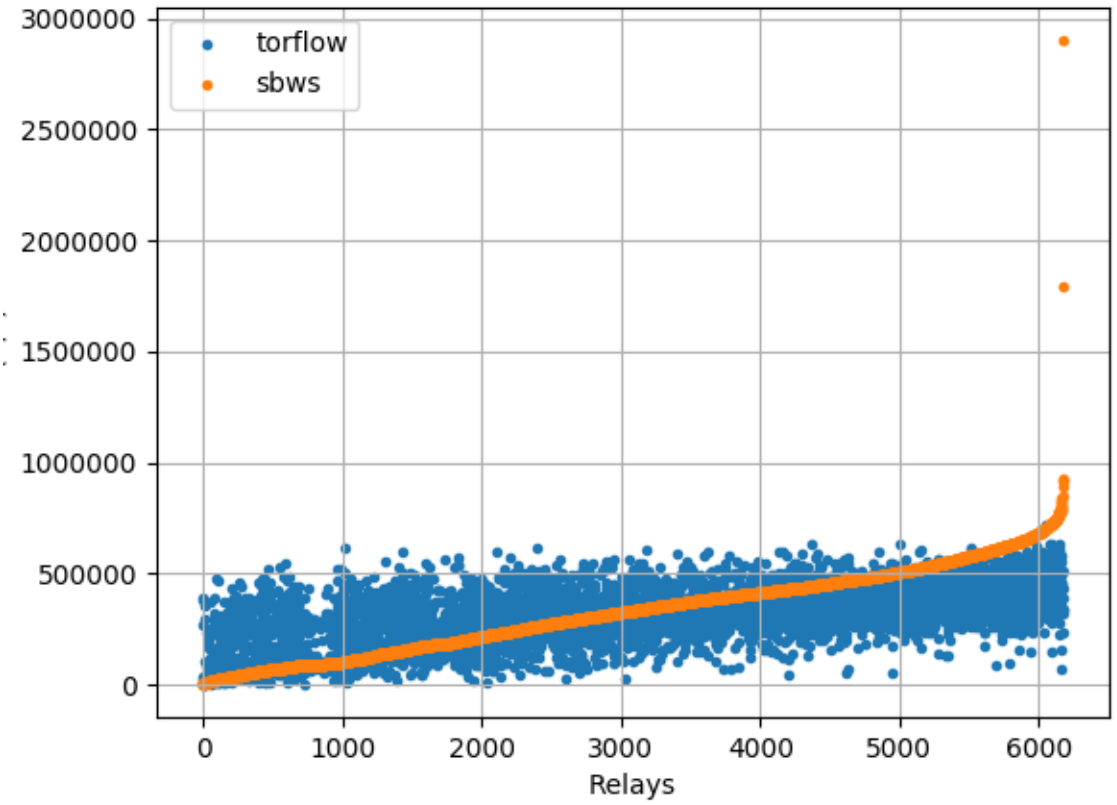
All these changes required lot of effort and are not optimal. It was the way we could correct and maintain 1.1.0 version. If a 2.0 version happens, we highly recommend re-design the data structures to use a database using a well maintained ORM library, which will avoid the limitations of json files, errors in data types conversions and which is optimized for the type of counting and statistics we aim to.

Note: Documentation about a possible version 2.0 and the steps to change the code from 1.X needs to be created.

2.14 Relays' bandwidth distribution

2.14.1 sbws raw measurements compared to Torflow measurements



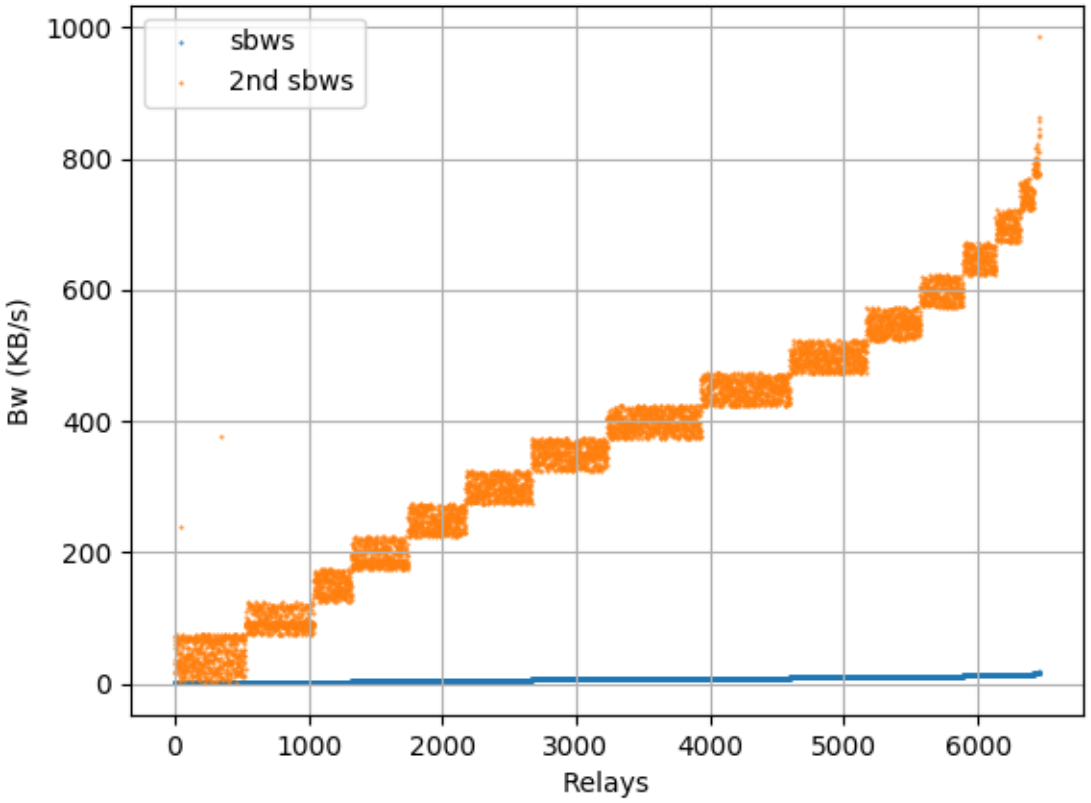


2.14.2 sbws linear scaling

Multiply each relay bandwidth by `7500/median`

See bandwidth_file_spec appendix B to know how about linear scaling.

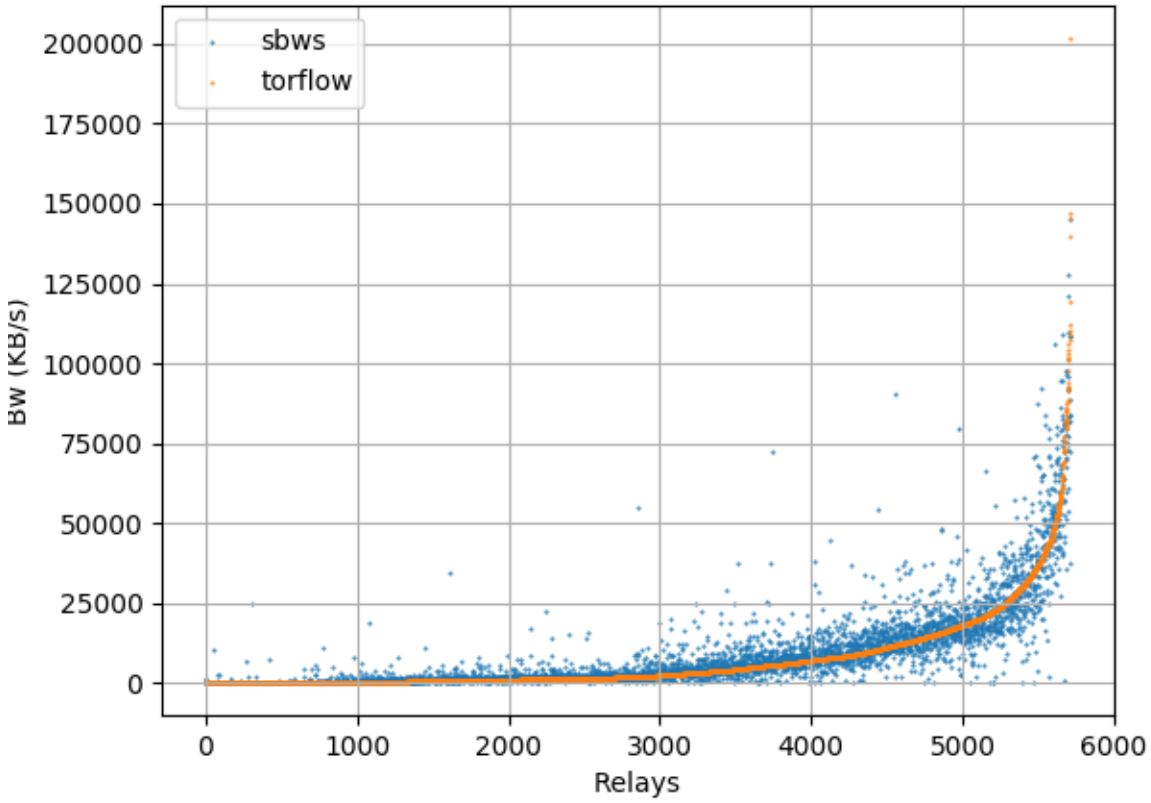
Code: `sbws.lib.v3bwfile.sbws_scale()`



2.14.3 sbws Torflow scaling

See bandwidth_file_spec appendix B to know how about torflow scaling.

Code: `sbws.lib.v3bwfile.torflow_scale()`



2.15 How bandwidth files are shown in the Tor network

2.15.1 Directory authorities' votes

moria, using Tor 0.3.5.7:

```
bandwidth-file-headers timestamp=1548181637
```

https://collector.torproject.org/recent/relay-descriptors/votes/

To appear in Tor v0.4.1.x:

```
bandwidth-file-digest sha256=0123456789012345678 9abcdefghijkl
```

https://gitlab.torproject.org/tpo/core/tor/-/issues/26698

2.15.2 Directory authorities' bandwidth file URL

To appear in Tor v0.4.1.x:

```
/tor/status-vote/next/bandwidth.z
```

https://gitlab.torproject.org/tpo/core/tor/-/issues/21377

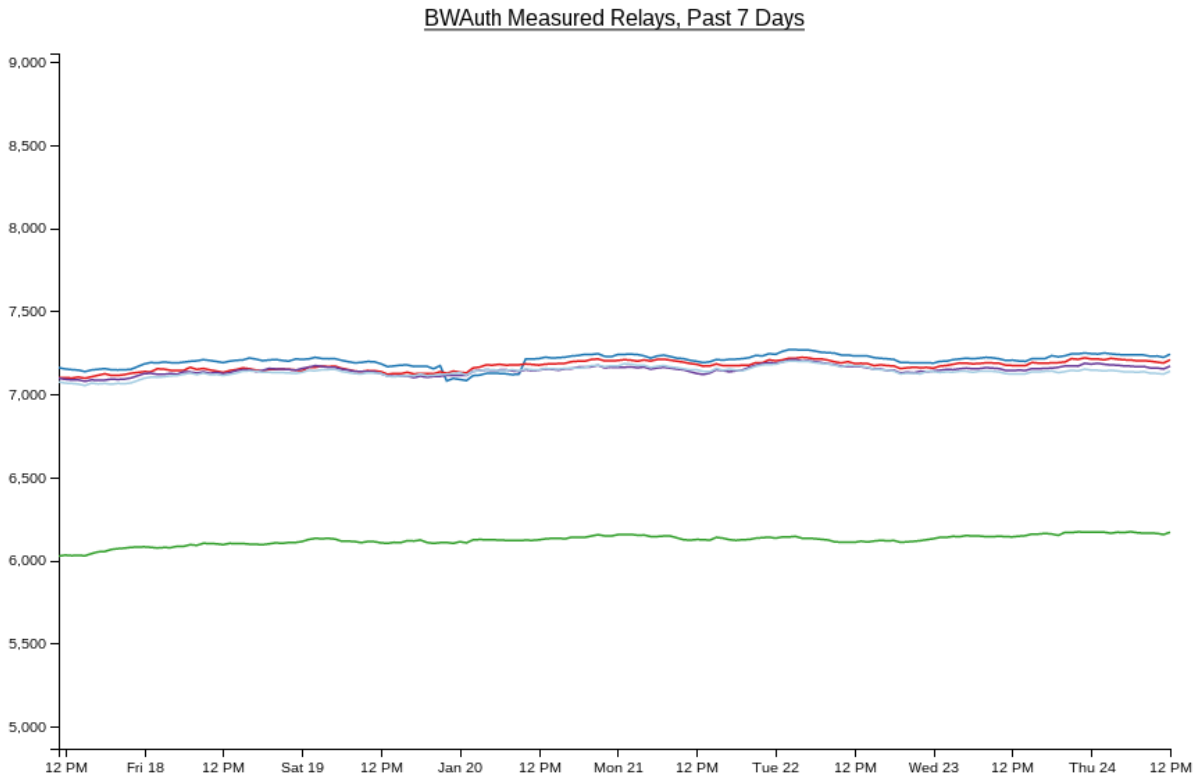
2.16 Bandwidth authorities in metrics

2.16.1 Current bandwidth authorities

https://metrics.torproject.org/rs.html

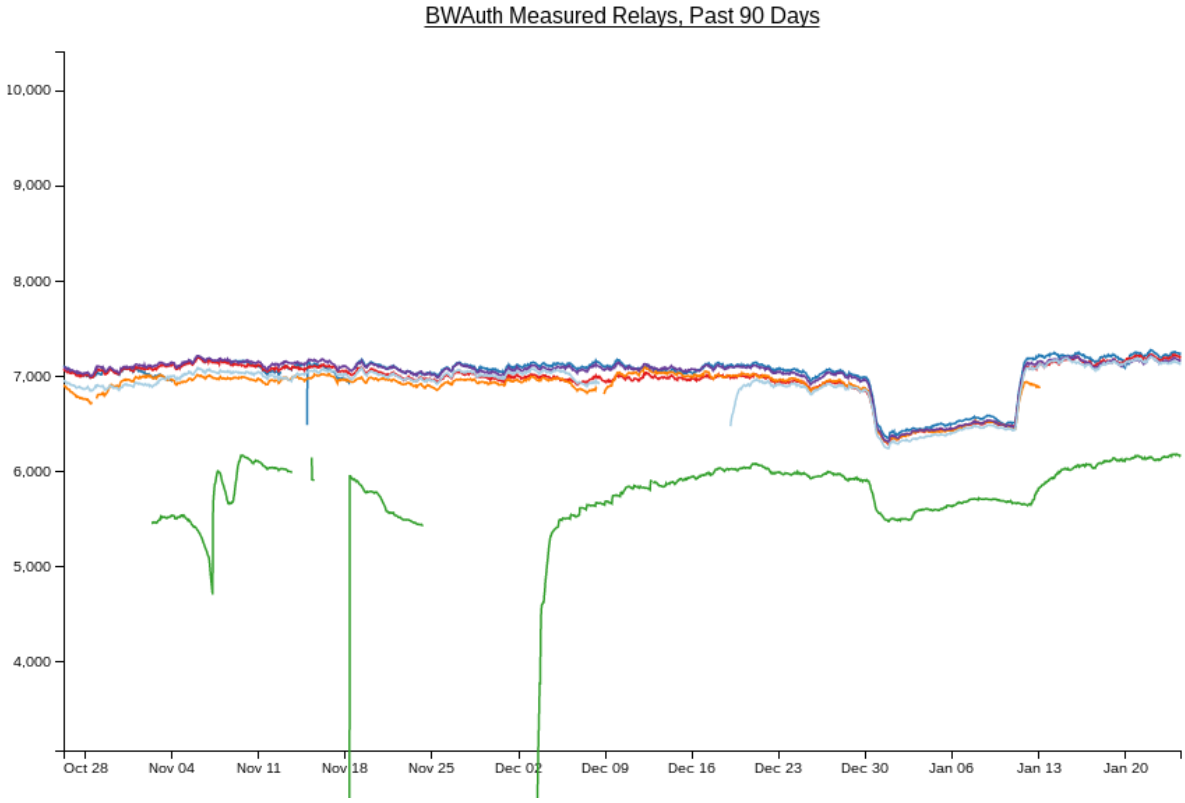
(flag:Authority)

2.16.2 Bandwidth Authorities - Measured Relays past 7 days



https://consensus-health.torproject.org/graphs.html

2.16.3 Bandwidth Authorities - Measured Relays past 90 days



https://consensus-health.torproject.org/graphs.html

2.17 Monitoring bandwidth changes in the Tor Network

2.17.1 Bandwidth authorities timeline

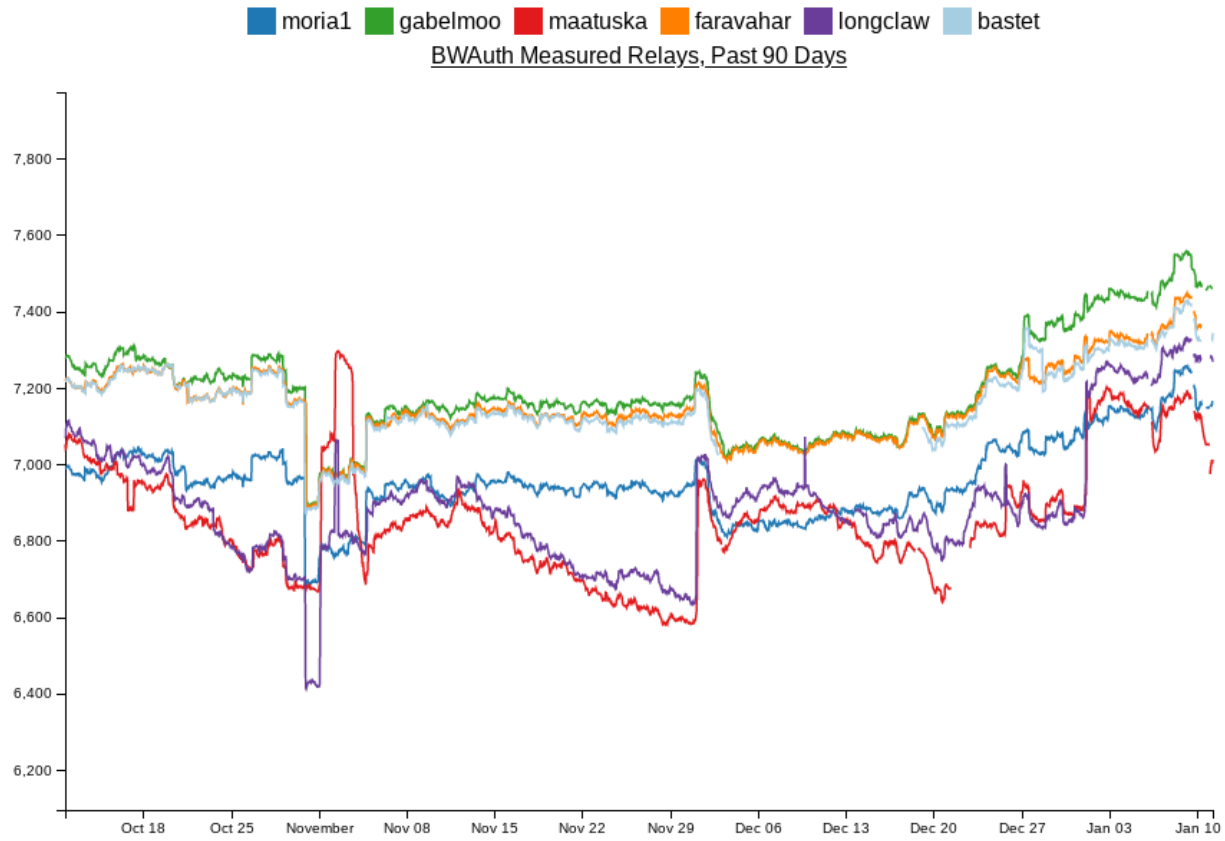
Events that can affect the data generated by the bwauths:

https://gitlab.torproject.org/tpo/network-health/sbws/-/wikis/bandwidth%20authorities%20timeline

This page might be moved to a different location.

2.17.2 Bwauths number of measured relays

It should be approximately equal for all bwauths.

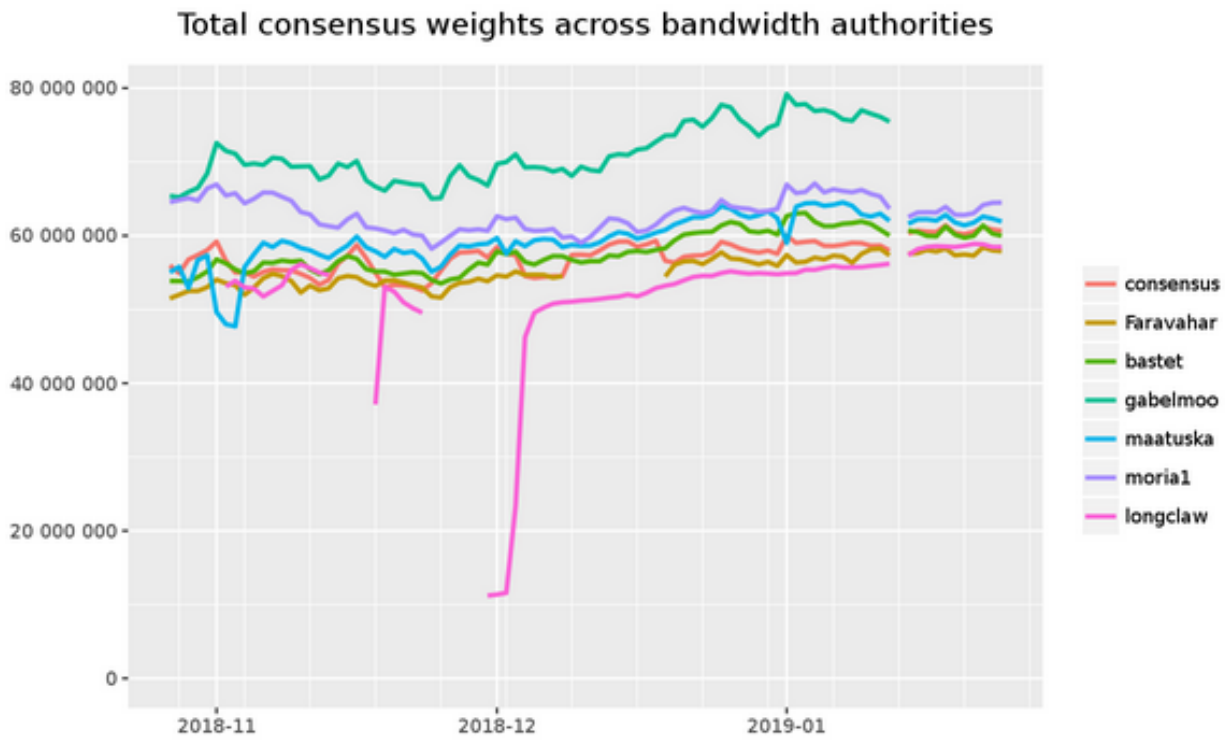


https://consensus-health.torproject.org/graphs.html#votedaboutgraphs

http://tgnv2pssfumdedyw.onion/graphs.html#votedaboutgraphs

2.17.3 Total consensus weights across bandwidth authorities

It should be approximately equal for all bwauths.



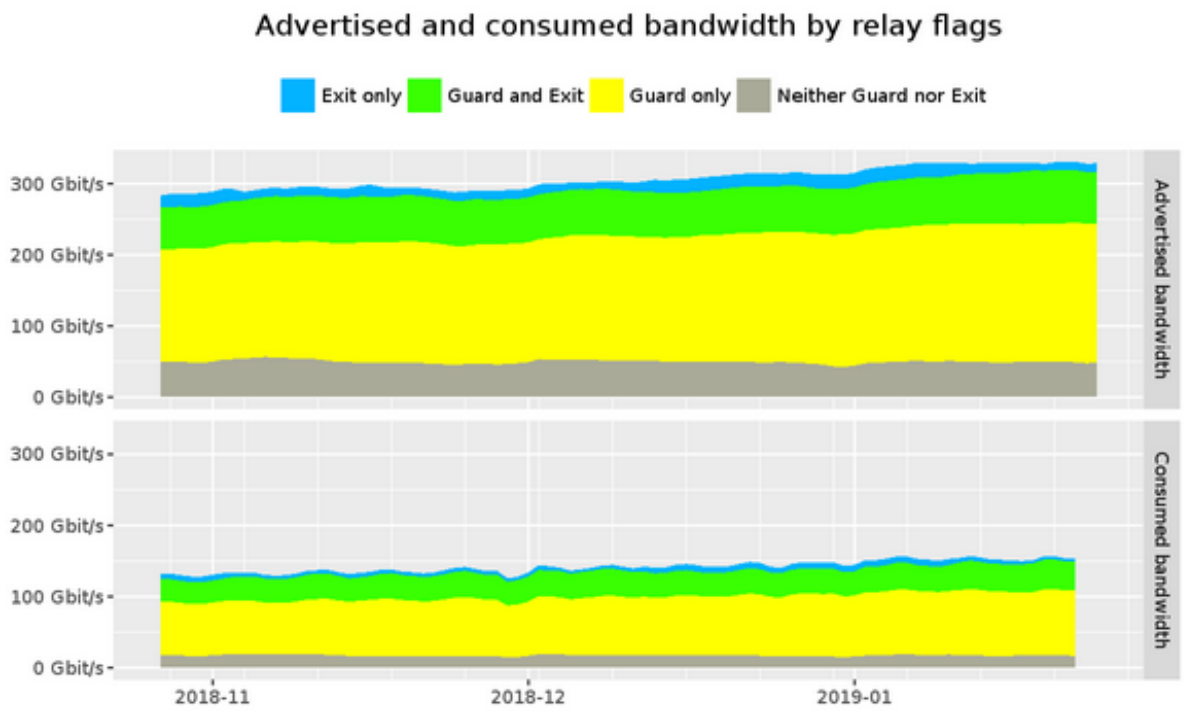
https://metrics.torproject.org/totalcw.html

2.17.4 Not measured relays and descriptors and consensus updates

Run the tool https://gitlab.torproject.org/juga/bwauthealth.

2.17.5 Total bandwidth

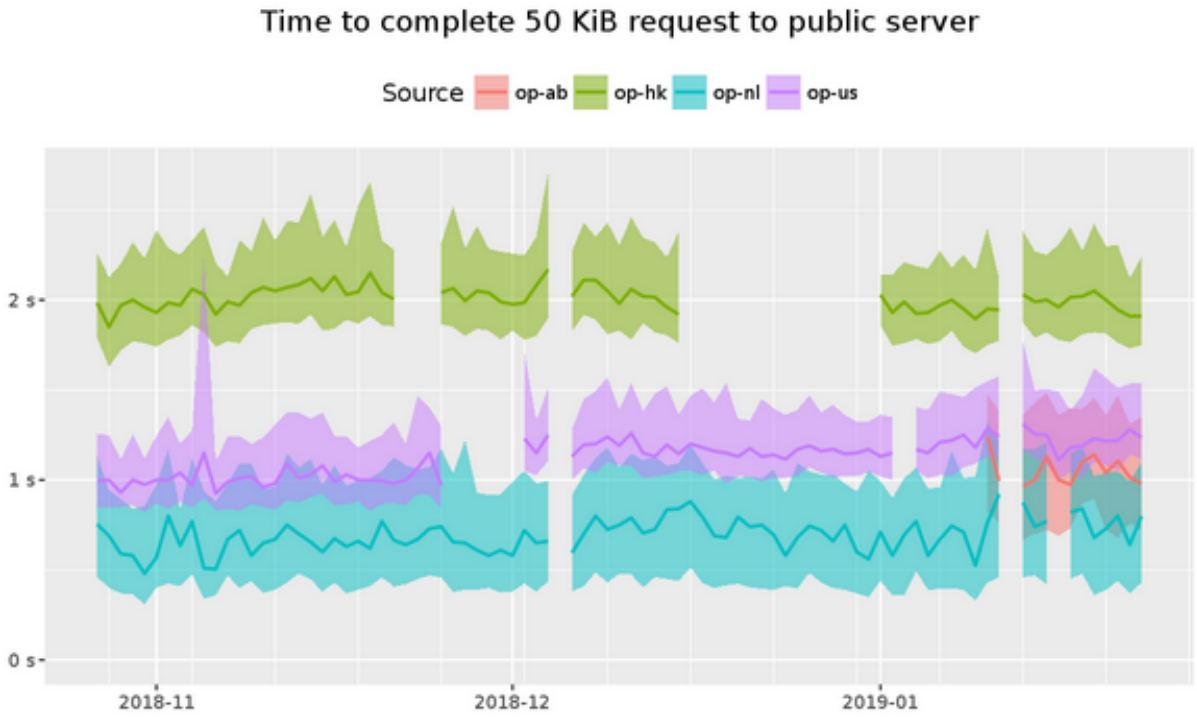
Should not decrease.



https://metrics.torproject.org/bandwidth-flags.html

2.17.6 Time to download a file

Should not increase.



https://metrics.torproject.org/torperf.html

2.18 Roadmap

2.18.1 Release 1.0.0

Autum 2018 - Minimal Viable Product (MVP)

2.18.2 Release 1.1.0

TBD

2.19 Glossary

directory authority a special-purpose relay that maintains a list of currently-running relays and periodically publishes a consensus together with the other directory authorities.[1]

bandwidth authority A *directory authority* that runs a *scanner* and a *generator* or obtain *bandwidth list file* s from a *generator*.

scanner Term to refer to the process that measures the relays' bandwidth. It is also called *generator* when it is the same tool that is used to generate *bandwidth list file* s.

generator Term to refer to the tool that generates the *bandwidth list file* s. Often used as a synomym for *scanner*.

bandwidth list file The file generated by *generator* s that is read by the *directory authority* s and included in their votes. See bandwidth-file-spec.txt to know about the file specification.

sbws scanner The `sbws` command used to run `sbws` as a *scanner*.

sbws generate The `sbws` command used to run `sbws` as a *generator*.

v3bw file The term used by `sbws` to refer to *bandwidth list file* v1.1.0.

Listing 2.4: A v3bw file

```
1524159868
version=0.1.0
node_id=$1BA71540E05D18401B65B553C35DA71992B9E488 bw=6941170 nick=exit2 rtt=20␣
↪time=1524107856
node_id=$189442066BEF15F777738E4E063B7BE0285EA0D9 bw=6855121 nick=exit3 rtt=19␣
↪time=1524107855
node_id=$076697F1272A92110AB82226699E62C4EFD49766 bw=6810514 nick=relay4 rtt=20␣
↪time=1524107855
node_id=$57BD9518CCC40874D969F0784922EF8B89EB9707 bw=6693692 nick=relay7 rtt=20␣
↪time=1524107837
node_id=$B5B33BCBC8C779BFE7B319E0CC3EA6E52EA355EA bw=6653275 nick=relay3 rtt=38␣
↪time=1524107847
node_id=$514326DD0EA15A41F1E7840C421A06CCCB2E39FA bw=6614808 nick=exit1 rtt=20␣
↪time=1524107837
node_id=$4E5FBF937A4C1D4F9211780BF700E70E30004910 bw=6593946 nick=relay1 rtt=19␣
↪time=1524107855
```

(continues on next page)

[1] https://metrics.torproject.org/glossary.html

(continued from previous page)

```
node_id=$D17B78F14F66F9F29686B37A78B77F6AC17DCE92 bw=6483024 nick=relay6 rtt=24␣
↪time=1524107848
node_id=$883505B618A0F14EE6136F1451CD4F00760C105F bw=6257421 nick=relay5 rtt=20␣
↪time=1524107865
node_id=$654E99AF0EAFA05DCD576C8607F15F3B076C53C8 bw=6069373 nick=relay2 rtt=19␣
↪time=1524107860
```

destination The term used by `sbws` to refer to a Web server where the *scanner* request files to perform the bandwith measurements.

2.20 Frequently Asked Questions (FAQ)

See also:

Glossary.

2.20.1 How many hops are the circuits used to perform the measurements?

Two hops.

2.20.2 How are relays selected to be measured?

The *sbws scanner* periodically refreshes its idea for what relays should be measured next. It prioritizese the measurement of relays that do not have recent results. In this way, relays that have just joined the network or have just come back online after a many-day period of being offline will be measured before relays that have been online constantly.

2.20.3 How do sbws scanner results end up in the consensus?

The *sbws scanner* runs continuously to gather fresh data.

The *sbws generate* command takes the fresh data and generates a *v3bw file*.

The Tor *directory authority* parses the v3bw file and includes bandwidth information in its vote.

The authorities take the low-median of the bandwidths for each relay from all of the *bandwidth authorities* and use that in the consensus.

2.20.4 Does sbws need any open ports?

No.

2.20.5 How much bandwidth will the sbws scanner use?

Todo: answer this

2.20.6 How much bandwidth will the webserver use?

Todo: answer this

2.20.7 Should I run my own webserver? Use a CDN? Something else?

It's up to you. Sbws is very flexible.

Todo: better answer.

Proposals:

2.21 Switching from helpers to HTTP(S)

Author Matt Traudt

Date 25 April 2018

Last Update 25 April 2018

Status Draft

2.21.1 Some Problems with the sbws helper concept

- Twice as many things to keep up to date
- Finding helper operators
- Trusting helper operators
- Home-grown protocol ("but it's totally secure enough for our needs guys, trust me. And it's totally implemented correctly too.")

2.21.2 Ways HTTP(S) is less terrible

- Standard protocols with established security properties
- Easier to "set and forget"
- **More flexible in deployment set ups**
  - TLS could be optional, but if present, could allow the web server to be far away from any Tor relay
  - CDNs could be used . . . maybe (no promises)
  - Measurements could still be done 1 relay at a time if there's a way to specify that the web server should be considered right next to a specific relay(s)

2.21.3 Challenges

Measuring 2 relays at a time

In the current design, it's easy to see and believe that sbws only measures one relay at a time.

In an sbws deployment that uses HTTP(S) servers far away from any Tor relay, it's harder or impossible. So we need to measure more than one relay at once. That means we need to come up with a way to select a pair of relays (where one is an exit, most likely) such that one won't significantly impact the results for the other.

Idea 1:

1. Pick a first hop relay.
2. Collect all exits that have a consensus weight equal or higher to the first hop relay. If there are none, collect the single fastest or select the `X` fastest.
3. Pick one of the collected exits in a weighted random fashion based on their consensus weights

Idea 2:

Same as the first, but consider all exits not just the ones faster than the first hop relay.

Supporting many variations on HTTP(S)

By this I mean I would love to support all of the following.

- sbws scanner option for using an HTTPS CDN across all exits in the network
- sbws scanner option for using a specific HTTPS webserver across all exits in the network (might not be any different than the previous item)
- sbws scanner option for using a specific HTTP(S) webserver across a specific list of relays (which may or may not have the Exit flag)
- in the cases where TLS is used, optional (enabled by default) verification the certificate is valid or at least pinned
- in the cases where TLS is used, the optional use of client certificates for identification

2.21.4 Proposals

Replace the concepts of "helpers" and "helper relays" and "sbws servers" with "SOMETHING". "Measurement methods"? "Avenues"? "Destinations"? I'll call them destinations for now.

Configuration

Replace `[helpers]` with `[destinations]`. If you don't remember what this section is for, it's for enabling/disabling various helpers (or, now destinations) without removing their config details.

```
[destinations]
cloudflare_cdn = on
pastly_relay = on
foobar = off
```

Replace `[helpers.foo]` with `[destinations.foo]`. If you don't remember what these are for, they are sections for each enabled destination that specify more specific configuration options for them. In the helper relay world, they had the relay fingerprint, sbws server host and port, and the password to give the sbws server.

If there is a combination of options that doesn't make sense, then sbws should fail to start.

If no destinations are configured, sbws should fail to start.

Sbws should run reachability tests on each destination on startup and then periodically and make sure they are usable as configured. It should only use destinations that are usable. If none are usable, it should sleep for a while and test for usability again later.

Available keys in a `[destinations.foo]` section:

- `relays`: a comma-separated list of relay fingerprints that can be used when using this destination. If unspecified, use all relays with the Exit flag. If specified, at least one relay must be usable. If it isn't, the destination should be considered unusable. (optional)
- `relay_section_method`: one of `uniform_random` or `bw_weighted_random`, defaulting to whichever is a sane default (optional)
- `url`: an HTTP or HTTPS URL for the bandwidth file to download. The URL's hostname must not be resolved locally; instead, it should be left up to the exit relay to resolve. If the URL does not contain a path, it defaults to `/sbws.bin`. (required)
- `weight`: when choosing between which destination to use for the next measurement, give this destination the specified weight. If not given, defaults to 100. Note how if no destinations have a weight value, they are chosen uniformly at random. (optional)

If protocol is https, these additional keys are available in a `[destionations.foo]`. It is a fatal configuration error to specify any of these if protocol is http.

- `client_cert`: path to certificate to provide to the server. If none provided, server is only usable if it doesn't require client authentication. If provided and file doesn't exist, it is a fatal configuration error. If provided and the server doesn't accept it, the destination is unusable. (optional)
- `verify_server_cert`: either a boolean or a path to a file. If yes (the default), the server's certificate must be trusted (as determined by the local machine's configuration outside of sbws). If no, do no verification of the certificate at all. If a path to a file and the file does not exist, it is a fatal configuration error. Otherwise, the certificate the server users must be present in the file pointed to by this option. (optional)

Example: CDN

Relays are not specified because we want to choose from all exits in the network.

This CDN provides `/sbws.bin` so we are allowed to leave off the file part.

HTTPS for the protocol, and no further HTTPS options because this CDN has a widely-trusted certificate and doesn't care about only allowing our sbws scanners to download files.

```
[destinations.cloudflare]
url = https://sbwsrocks.cdn.cloudflare.com/
```

Example: Private Local Destination

Here, an authority has decided he doesn't want to trust anyone but themself. They are running 2 relays on the same machine as a webserver that only they will use.

This authority chooses to use a client TLS certificate to identify their scanner(s), so their webserver must use HTTPS.

On their webserver they generate a self-signed certificate. On the sbws scanner side, they *could* choose to assume everything will be okay and his server will not change certificates. But they're paranoid, so they get a copy of the server's certificate and store it in a local file.

Todo: What file format?

```
[destionations.secure_bwauth]
relays = AAAA...AAAA, BBBB...BBBB
relay_section_method = uniform_random
url = https://33.33.33.33:4433/sbws.bin
client_cert = ${paths:sbws_home}/secure_bwauth_scanner.cert
verify_server_cert = ${paths:sbws_home}/secure_bwauth_server.cert
```

Example: "Borrow" bandwidth from unsuspecting mirrors

This could be considered unethical and therefore a terrible non-starter idea.

It's also a cool thing that I think is technically possible.

Pick a Linux distro that provides ISOs or packages over an HTTP(S) server. Ideally many servers under a single DNS name that rotates. (Maybe even one that is geo-aware to give you a close mirror to where you're resolving the name.)

Then just find a file big enough to service all of our possible request sizes, and add it to the config.

```
[destination.unsuspecting_linux]
url = http://examplelinux.net/archive/isos/1.2.3/examplelinux-amd64-gnome-destkop.iso
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Bibliography

[DIST] https://git-scm.com/book/en/v2/Distributed-Git-Contributing-to-a-Project

[MERG] https://www.atlassian.com/git/tutorials/merging-vs-rebasing

[REVI] https://doc.sagemath.org/html/en/developer/reviewer_checklist.html

[FUNC] https://medium.com/@rohanrony/functional-programming-in-python-1-lambda-map-filter-reduce-zip-8739ea144186

Python Module Index

S

Index

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W