Work in progress notice

This page is going to be a work in progress until late 2018. The only reason we're putting this page here is to facilitate work on this document. Handle with care!

Change log:

- 2018-03-31: Add Users section with Relay users subsection.
- 2018-04-21: Complete Users section by adding Bridge users subsection.
- 2018-04-23: Add Traffic section with one subsection.
- 2018-04-23: Add Performance section with single subsection.
- 2018-04-24: Add Applications section with single subsection.
- 2018-04-27: Add Servers section with two subsections.
- 2018-05-03: Add Traffic section with three subsections.
- 2018-05-04: Add Onion Services section with single subsection.

Reproducible Metrics

The graphs and tables on Tor Metrics are the result of aggregating data obtained from several points in the Tor network. Some of these aggregations are straightforward, but some are not.

We want to make the graphs and tables on this site easier to access and reproduce, so on this page, we specify how you can reproduce the data behind them to create your own. We also provide background for some of the design decisions behind our aggregations and link to technical reports (https://research.torproject.org/techreports.html) and other additional information.

This page is a living document that reflects the latest changes to graphs and tables on Tor Metrics. Whenever we create new aggregations or visualizations, we may write down our thoughts in technical reports; but, if we later expand or change a statistic, we don't update the original technical reports. Instead, we update the specification here.

While we may refer to technical reports for additional details, we do not assume their knowledge in order to make sense of the specifications here. Knowledge of our source code is not needed, either.

Users

The number of Tor users is one of our most important statistics. It is vital for us to know how many people use the Tor network on a daily basis, whether they connect via relays or bridges, from which countries they connect, what transports they use, and whether they connect via IPv4 or IPv6.

Due to the nature of Tor being an anonymity network, we cannot collect identifying data to learn the number of users. That is why we actually don't count users, but we count requests to the directories or bridges that clients make periodically to update their list of relays and estimate user numbers indirectly from there.

The result is an average number of concurrent users, estimated from data collected over a day. We can't say how many distinct users there are. That is, we can't say whether the same set of users stays connected over the whole day, or whether that set leaves after a few hours and a new set of users arrives. However, the main interest is finding out if usage changes, for which it is not critical to estimate exact absolute user numbers.

Relay users

Relay users are users that connect directly to a relay in order to connect to the Tor network as opposed to bridge users that connect to a bridge as entry point into the Tor network. Many steps here are similar to the steps for estimating bridge users, which are specified further down below.

The following description applies to the following graph and tables:

- Relay users graph (/userstats-relay-country.html)
- Top-10 countries by relay users table (/userstats-relay-table.html)
- Top-10 countries by possible censorship events table (/userstats-censorship-events.html)

Step 1: Parse consensuses to learn which relays have been running

Obtain consensuses from CollecTor (/collector.html#type-network-status-consensus-3). Refer to the Tor directory protocol, version 3 (https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt) for details on the descriptor format.

From each consensus, parse the `"valid-after"` and `"fresh-until"` times from the header section.

From each consensus entry, parse the base64-encoded relay fingerprint from the `"r"` line. Also parse the relay flags from the `"s"` line. If there is no `"Running"` flag, skip this entry. (Consensuses with consensus method 4, introduced in 2008, or later do not list non-running relays, so that checking relay flags in recent consensuses is mostly done as a precaution without actual effect on the parsed data.)

Example

Original consensus (excerpt):

```
valid-after 2018-03-14 12:00:00
fresh-until 2018-03-14 13:00:00
r seele AAoQ1DAR6kkoo19hBAX5K0QztNw +NM1/rB7bUT58qIXQUyuUUIUswE 2018-03-14 04:48:26 67.
s Fast Running Stable V2Dir Valid
r Unnamed AAwffNL+oHO5EdyUoWAOwvEX3ws uslFs811AQoZpso0Yh+wjXqSv1U 2018-03-14 11:27:53 1
s Fast Running V2Dir Valid
```

Parsed consensus contents:

Description	Content
Valid-after time	2018-03-14 12:00:00
Fresh-until time	2018-03-14 13:00:00
Fingerprints of running relays (converted to hex)	000A10D43011EA4928A35F610405F92B4433B4DC
	000C1F7CD2FEA073B911DC94A1600EC2F117DF0B
	…

Step 2: Parse relay extra-info descriptors to learn relevant statistics reported by relays

Also obtain relay extra-info descriptors from CollecTor (/collector.html#type-extra-info). As above, refer to the Tor directory protocol, version 3 (https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt) for details on the descriptor format.

Parse the relay fingerprint from the `"extra-info"` line and the descriptor publication time from the `"published"` line.

Parse the `"dirreq-write-history"` line containing written bytes spent on answering directory requests. If the contained statistics end time is more than 1 week older than the descriptor publication time in the `"published"` line, skip this line to avoid including statistics in the aggregation that have very likely been reported and processed before. If a statistics interval spans more than 1 UTC date, split observations to the covered UTC dates by assuming a linear distribution of observations.

Parse the `"dirreq-stats-end"` and `"dirreq-v3-reqs"` lines containing directory-request statistics. If the statistics end time in the `"dirreq-stats-end"` line is more than 1 week older than the descriptor publication time in the `"published"` line, skip these directory request statistics for the same reason as given above: to avoid including statistics in the aggregation that have very likely been reported and processed before. Also skip statistics with an interval length other than 1 day. Parse successful requests by country from the `"dirreq-v3-reqs"` line. From each number, subtract `4` to undo the binning operation that has been applied by the relay. Discard the resulting number if it's zero or negative. Split observations to the covered UTC dates by assuming a linear distribution of observations.

Example

Original extra-info descriptor (excerpt):

```
extra-info deathsfollyash CD54F0946A30E366A8392D04D7ED684A121069FC
published 2018-03-15 05:58:56
dirreq-write-history 2018-03-15 05:53:06 (14400 s) 1935625216,1916100608,2856720384,264
dirreq-stats-end 2018-03-14 20:28:43 (86400 s)
dirreq-v3-reqs us=4000,de=2304,ru=2304,ua=1368,fr=808,gb=568,ca=536,nl=448,br=408,it=3
```

Parsed extra-info descriptor contents:

Description	Content
Relay fingerprint	CD54F0946A30E366A8392D04D7ED684A121069FC
Descriptor publication time	2018-03-15 05:58:56
Written directory bytes statistics	1935625216 bytes between 2018-03-14 05:53:06 and 2018-03-14 09:53:06
	1916100608 bytes between 2018-03-14 09:53:06 and 2018-03-14 13:53:06
	2856720384 bytes between 2018-03-14 13:53:06 and 2018-03-14 17:53:06
	2648868864 bytes between 2018-03-14 17:53:06 and 2018-03-14 21:53:06
	815703725 = 1542702080 * 7614 / 14400 bytes between 2018-03-14 21:53:06 and 2018-03-15 00:00:00
	726998355 = 1542702080 * 6786 / 14400 bytes between 2018-03-15 00:00:00 and 2018-03-15 01:53:06
	1359456256 bytes between 2018-03-15 01:53:06 and 2018-03-15 05:53:06
Successful directory requests by country	586.31 = (4000 - 4) * 12677 / 86400 requests from the United States ("us") between 2018-03-13 20:28:43 and 2018-03-14 00:00:00
	3409.69 = (4000 - 4) * 73723 / 86400 requests from the United States ("us") between 2018-03-14 00:00:00 and 2018-03-14 20:28:43
	337.47 = (2304 - 4) * 12677 / 86400 requests from Germany ("de") between 2018-03-13 20:28:43 and 2018-03-14 00:00:00
	1962.53 = (2304 - 4) * 73723 / 86400 requests from Germany ("de") between 2018-03-14 00:00:00 and 2018-03-14 20:28:43
	…

Step 3: Estimate fraction of reported directory-request statistics

The next step after parsing descriptors is to estimate the fraction of reported directory-request statistics on a given day. This fraction, a value between *0%* and *100%*, will be used in the next step to extrapolate observed request numbers to expected network totals. For further background on the following calculation method, refer to the technical report titled "Counting daily bridge users" (https://research.torproject.org/techreports/counting-daily-bridge-users-2012-10-24.pdf) which also applies to relay users. In the following, we're using the term server instead of relay or bridge, because the estimation method is exactly the same for relays and bridges.

For each day in the time period, compute five variables:

- Compute *n(N)* as the total server uptime on a given day, that is, the sum of all server uptimes on that day. This is the sum of all intervals between `"valid-after" and "fresh-until"`, multiplied by the contained running servers, for all consensuses with a valid-after time on a given day. A more intuitive interpretation of this variable is the average number of running servers—however, that interpretation only works as long as fresh consensuses are present for all hours of a day.
- Compute *n(H)* as the sum of intervals for which servers have reported written directory bytes on a given day.
- Compute *n(R\H)* as the number of seconds for which responses have been reported but no written directory bytes. This fraction is not computed by comparing all reported intervals. Instead, this fraction is determined by summing up all interval lengths and then subtracting the written directory bytes interval length from the directory response interval length. Negative results are discarded.
- Compute *h(H)* as the total number of written directory bytes on a given day.
- Compute *h(R^H)* as the number of written directory bytes for the fraction of time when a server was reporting both written directory bytes and directory responses. As above, this fraction is not computed by comparing all reported intervals. Instead, this fraction is determined by first summing up all interval lengths and then computing the minimum of both sums divided by the sum of reported written directory bytes.

From these variables, compute the estimated fraction of reported directory-request statistics using the following formula:

```
       h(R^H) * n(H) + h(H) * n(R\H)
frac = ----------------------------- where h(H) * n(N) > 0.
                h(H) * n(N)
```

Example

Variable values for 2018-03-14:

Variable	Value
n(N)	6225 days
n(H)	6343 days
n(R\H)	18 days
h(H)	19.0 TiB
h(R^H)	16.9 TiB
frac	91.2%

Remarks:

- The value of *n(H)* is comparable to the value of *n(N)*. Typically, one would expect the latter to be smaller, as some servers are not reporting statistics and others are reporting statistics only for a fraction of their uptime on this day. In this case, though, some servers apparently reported statistics for times when they have not been listed as running in the consensus.
- The value of *n(R\H)* is almost negligible compared to *n(N)* and *n(H)*. Almost all servers report either both statistics or none of them.
- The value of *h(H)* is comparable to the value of *h(R^H)*. Again, this indicates that servers reported either both or none of the statistics.
- The estimated fraction of reported directory-request statistics, frac, is 91.2%. As an example, if all servers together, including those that did not report statistics, had handled 1000 requests from a given country, we'd expect statistics to include ≈ 912 of them.

Step 4: Compute estimated relay users per country

With the estimated fraction of reported directory-request statistics from the previous step it is now possible to compute estimates for relay users. Similar to the previous step, the same approach described here also applies to estimating bridge users by country, transport, or IP version as described further down below.

First compute *r(R)* as the sum of reported successful directory requests from a given country on a given day. (Obviously, this approach also works with *r(R)* being the sum of requests from *all* countries or from any other subset of countries, if this is of interest.)

Estimate the number of clients per country and day using the following formula:

```
r(N) = r(R) / (frac * 10)
```

Skip dates where *frac* is smaller than 10% and hence too low for a robust estimate, or where *frac* is greater than 100%, which would indicate an issue in the previous step.

The number *10* in the denominator comes from the number of directory requests that an average client makes every day. A client that is connected 24/7 makes about 15 requests per day, but not all clients are connected 24/7, so we picked the number 10 for the average client. We simply divide directory requests by 10 and consider the result as the number of users. Another way of looking at it, is that we assume that each request represents a client that stays online for one tenth of a day, so 2 hours and 24 minutes.

Example

Variable values for 2018-03-14 and for requests from the United Kingdom ("gb"):

Variable	Value

r(R)	570824 requests
frac	92.1% (see previous step)
r(N)	62617 users

Step 5: Compute ranges of expected clients per day to detect potential censorship events

As last step in reproducing relay user numbers, compute ranges of expected clients per day to detect potential censorship events. For further details on the detection method, refer to the technical report titled "An anomaly-based censorship-detection system for Tor" (https://research.torproject.org/techreports/detector-2011-09-09.pdf). Unlike the previous two steps, this step only applies to relay users, not to bridge users.

1. Start by finding the 50 largest countries by estimated relay users, excluding `"??"` , on the last day in the data set. (Note that, as new data gets available, the set of 50 largest countries may change, too, affecting ranges for the entire data set.)
2. For each of these largest countries and for each date in the data set, compute the ratio R_ij = C_ij / C_(i-7)j of estimated users on any given date divided by estimated users 1 week earlier. Exclude ratios for which there are no user estimates from 1 week earlier, or where that estimate is 0.
3. For the computed ratios on each date, remove outliers that fall outside four inter-quartile ranges of the median, and remove dates with less than 8 ratios remaining.
4. For each date, fit a normal distribution to the remaining ratios.
5. For each date and country, compute a range of expected user numbers by using the fitted normal distribution and the ratio of estimated users divided by estimated users 1 week earlier as input. As previously, exclude ratios for which there are no user estimates from 1 week earlier, or where that estimate is 0. Compute the low and high ranges as:
   - low = max(0, NormalDistribution(mu, standard deviation, P = 0.0001) * PoissonDistribution(lambda = C_(i-7)j, P = 0.0001))
   - high = NormalDistribution(mu, standard deviation, P = 0.9999) * PoissonDistribution(lambda = C_(i-7)j, P = 0.9999)

Example

Variable values for 2018-03-14:

Variable	Value
Largest 50 countries on 2018-03-24	382898 users from the United States ("us")
	339797 users from the United Arab Emirates ("ae")
	… (omitted 46)
	4250 users from Oman ("om")
	4145 users from Portugal ("pt")
Ratios *R_ij* on 2018-03-14	1.071 = 415446 / 388078 for the United States ("us")
	0.969 = 302161 / 311726 for the United Arab Emirates ("ae")
	… (omitted 46)
	0.971 = 4650 / 4790 for Oman ("om")
	1.000 = 4233 / 4235 for Portugal ("pt")
Median and four inter-quartile ranges of ratios	0.976 ± 0.287

Ratios (with outliers struck through)	~~0.513~~, 0.705, 0.771, 0.821, 0.854, 0.868, 0.869, 0.878, 0.886, 0.893, 0.906, 0.920, 0.929, 0.935, 0.936, 0.943, 0.951, 0.963, 0.966, 0.967, 0.969, 0.971, 0.972, 0.975, 0.975, 0.976, 0.978, 0.980, 0.987, 0.988, 0.989, 0.989, 0.989, 0.990, 0.994, 0.998, 1.000, 1.001, 1.016, 1.025, 1.029, 1.032, 1.033, 1.034, 1.065, 1.071, 1.083, 1.088, 1.116, ~~2.120~~
Parameters of fitted normal distribution	mu = 0.964, standard deviation = 0.078
Computed low and high ranges for the United Kingdom ("gb")	low = 43055, high = 82396
Actual user number estimate for the United Kingdom ("gb")	62617

Bridge users

Bridge users are users that connect to a bridge as entry point into the Tor network as opposed to relay users that connect directly to a relay. Many steps here are similar to the steps for estimating relay users, which are specified above.

The following description applies to the following graphs and table:

- Bridge users by country graph (/userstats-bridge-country.html)
- Bridge users by transport graph (/userstats-bridge-transport.html)
- Bridge users by country and transport graph (/userstats-bridge-combined.html)
- Bridge users by IP version graph (/userstats-bridge-version.html)
- Top-10 countries by bridge users table (/userstats-bridge-table.html)

Step 1: Parse bridge network statuses to learn which bridges have been running

Obtain bridge network statuses from CollecTor (/collector.html#type-bridge-network-status). Refer to the Tor bridge descriptors page (/bridge-descriptors.html) for details on the descriptor format.

From each status, parse the `"published"` time from the header section.

From each status entry, parse the base64-encoded hashed bridge fingerprint from the `"r"` line. Also parse the relay flags from the `"s"` line. If there is no `"Running"` flag, skip this entry.

As opposed to relay consensuses, there are no `"valid-after"` or `"fresh-until"` times in the header of bridge network statuses. To unify processing, we use the publication hour as valid-after time and one hour later as fresh-until time. If we process multiple statuses published in the same hour, we take the union of contained running bridges as running bridges in that hour.

Example

Sanitized bridge network status (excerpt):

```
published 2018-03-14 12:04:07
r Unnamed ACzV/+Qobny0/Tk7Jodk8cP+ME4 FRJ7hToK0RTUTfkXc39JfNEfjtM 2018-03-14 09:03:26
s Fast HSDir Running Stable V2Dir Valid
r Unnamed ADXqKmHijTlfCArKIkRTlJDnCVA 221u05jm8/F+n83SjTdqVrzAv+k 2018-03-14 09:41:14
s Fast HSDir Running Stable V2Dir Valid
```

Parsed status contents:

Description	Content

Publication time	2018-03-14 12:04:07
Assumed valid-after time	2018-03-14 12:00:00
Assumed fresh-until time	2018-03-14 13:00:00
Hashed fingerprints of running bridges (converted to hex)	002CD5FFE4286E7CB4FD393B268764F1C3FE304E
	0035EA2A61E28D395F080ACA2244539490E70950
	…

Step 2: Parse bridge extra-info descriptors to learn relevant statistics reported by bridges

Also obtain bridge extra-info descriptors from CollecTor (/collector.html#type-bridge-extra-info). As above, refer to the Tor bridge descriptors page (/bridge-descriptors.html) for details on the descriptor format.

Parse the hashed bridge fingerprint from the `"extra-info"` line and the descriptor publication time from the `"published"` line.

Parse the `"dirreq-write-history"` line containing written bytes spent on answering directory requests. If the contained statistics end time is more than 1 week older than the descriptor publication time in the `"published"` line, skip this line to avoid including statistics in the aggregation that have very likely been reported and processed before. If a statistics interval spans more than 1 UTC date, split observations to the covered UTC dates by assuming a linear distribution of observations.

Parse the `"dirreq-stats-end"` and `"dirreq-v3-resp"` lines containing directory-request statistics. If the statistics end time in the `"dirreq-stats-end"` line is more than 1 week older than the descriptor publication time in the `"published"` line, skip these directory request statistics for the same reason as given above: to avoid including statistics in the aggregation that have very likely been reported and processed before. Also skip statistics with an interval length other than 1 day. Parse successful requests from the `"ok"` part of the `"dirreq-v3-resp"` line. Subtract `4` to undo the binning operation that has been applied by the bridge. Discard the resulting number if it's zero or negative. Split observations to the covered UTC dates by assuming a linear distribution of observations.

Parse the `"bridge-ips"`, `"bridge-ip-versions"`, and `"bridge-ip-transports"` lines containing unique connecting IP addresses by country, IP version, and transport. From each number of unique IP addresses, subtract 4 to undo the binning operation that has been applied by the bridge. Discard the resulting number if it's zero or negative.

Example

Original bridge extra-info descriptor (excerpt):

```
extra-info Lisbeth D9C805C955CB124D188C0D44F271E9BE57DE2109
published 2018-03-14 22:05:35
dirreq-write-history 2018-03-14 16:09:04 (86400 s) 5937854464,5248079872,5660442624,55
dirreq-stats-end 2018-03-14 18:28:56 (86400 s)
dirreq-v3-resp ok=18584,not-enough-sigs=0,unavailable=0,not-found=0,not-modified=888,bu
bridge-ips ru=9464,us=5272,??=3808,tr=3408,ua=2200,by=1920,de=1888,gb=1688,ir=1448,ae=
bridge-ip-versions v4=42168,v6=5872
bridge-ip-transports <OR>=8,obfs4=48032
```

Parsed bridge extra-info descriptor contents:

Description	Content
Hashed bridge fingerprint	D9C805C955CB124D188C0D44F271E9BE57DE2109
Descriptor publication time	2018-03-14 22:05:35

Written directory bytes statistics	1941898330 = 5937854464 * 28256 / 86400 bytes between 2018-03-09 16:09:04 and 2018-03-10 00:00:00
	3995956134 = 5937854464 * 58144 / 86400 bytes between 2018-03-10 00:00:00 and 2018-03-10 16:09:04
	1716316491 = 5248079872 * 28256 / 86400 bytes between 2018-03-10 16:09:04 and 2018-03-11 00:00:00
	3531763381 = 5248079872 * 58144 / 86400 bytes between 2018-03-11 16:09:04 and 2018-03-11 16:09:04
	…
Successful directory requests	4271.68 = (18584 - 4) * 19864 / 86400 requests between 2018-03-13 18:28:56 and 2018-03-14 00:00:00
	14308.32 = (18584 - 4) * 66536 / 86400 requests between 2018-03-14 00:00:00 and 2018-03-14 18:28:56
Unique IP addresses by country	9460 = (9464 - 4) unique IP addresses from Russia ("ru")
	5268 = (5272 - 4) unique IP addresses from the United States ("us")
	…
	48068 unique IP addresses from all countries
Unique IP addresses by IP version	42164 = (42168 - 4) unique IPv4 addresses
	5868 = (5872 - 4) unique IPv6 addresses
Unique IP addresses by transport	4 = (8 - 4) unique IP addresses via the default onion-routing protocol
	48028 = (48032 - 4) unique IP addresses via the obfs4 protocol
	…

Step 3: Approximate directory requests by country, transport, and IP version

Bridges, unlike relays, do not report directory request numbers by country, transport, or IP version. However, bridges do report unique IP address counts by country, by transport, and by IP version. We approximate directory request numbers by multiplying the fraction of unique IP addresses from a given country, transport, or IP version with the total number of successful requests.

As a special case, we also approximate lower and upper bounds for directory requests by country *and* transport. This approximation is based on the fact that most bridges only provide a small number of transports. This allows us to combine unique IP address sets by country and by transport and obtain lower and upper bounds:

- We calculate the lower bound as `max(0, C(b) + T(b) − S(b))` using the following definitions: `C(b)` is the number of requests from a given country reported by bridge `b`; `T(b)` is the number of requests using a given transport reported by bridge `b`; and `S(b)` is the total numbers of requests reported by bridge `b`. Reasoning: If the sum `C(b) + T(b)` exceeds the total number of requests from all countries and transports `S(b)`, there must be requests from that country and transport. And if that is not the case, `0` is the lower limit.
- We calculate the upper bound as `min(C(b), T(b))` with the definitions from above. Reasoning: There cannot be more requests by country and transport than there are requests by either of the two numbers.

If a bridge does not report unique IP addresses by country, transport, or IP version, we attribute all requests to "??" which stands for Unknown Country, to the default onion-routing protocol, or to IPv4.

Example (continued from previous step)

Approximated directory request numbers for the time between 2018-03-13 18:28:56 and 2018-03-14 00:00:00 (excerpt):

Description	Content
Successful directory requests	4271.68 requests (as calculated in the previous step)
… by country	840.69 = (9464 - 4) / 48068 * 4271.68 requests from Russia ("ru")
	468.15 = (5272 - 4) / 48068 * 4271.68 requests from the United States ("us")
… by IP version	3749.82 = (42168 - 4) / 48032 * 4271.68 requests via IPv4
	521.86 = (5872 - 4) / 48032 * 4271.68 requests via IPv6
… by transport	0.36 = (8 - 4) / 48032 * 4271.68 requests via the default onion-routing protocol
	4271.32 = (48032 - 4) / 48032 * 4271.68 requests via the obfs4 protocol
… by country and transport	840.33 = max(0, 840.69 + 4271.32 - 4271.68) as lower bound for requests from Russia ("ru") via the obfs4 protocol
	840.69 = min(840.69, 4271.32) as upper bound for requests from Russia ("ru") via the obfs4 protocol

Step 4: Estimate fraction of reported directory-request statistics

The step for estimating the fraction of reported directory-request statistics is pretty much the same for bridges and for relays. This is why we refer to Step 3 of the Relay users description for this estimation.

Step 5: Compute estimated bridge users per country, transport, or IP version

Similar to the previous step, this step is equivalant for bridge users and relay users. We therefore refer to Step 4 of the Relay users description for transforming directory request numbers to user numbers.

Servers

Statistics on the number of servers relays and bridges were among the first to appear on Tor Metrics. These statistics have one thing in common: they use the number of running servers as their metric. Possible alternatives would be to use consensus weight fractions or guard/middle/exit probabilities as metrics, but we're not doing that yet. In the following, we describe how exactly we count servers.

Running relays

We start with statistics on the number of running relays in the network, broken down by criteria like assigned relay flags, self-reported tor version and operating system, or IPv6 capabilities.

The following description applies to the following graphs:

- Relays and bridges (just the relays part; for the bridges part see below) graph (/networksize.html)
- Relays by relay flag graph (/relayflags.html)
- Relays by tor version graph (/versions.html)
- Relays by platform graph (/platforms.html)
- Relays by IP version graph (/relays-ipv6.html)

Step 1: Parse consensuses

Obtain consensuses from CollecTor (/collector.html#type-network-status-consensus-3). Refer to the Tor directory protocol, version 3 (https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt) for details on the descriptor format.

Parse and memorize the `"valid-after"` time from the consensus header. We use this UTC timestamp to uniquely identify the consensus while processing, and to later aggregate by the UTC date of this UTC timestamp.

Repeat the following steps for each consensus entry:

- Server descriptor digest: Parse the server descriptor digest from the "r" line. This is only needed for statistics based on relay server descriptor contents.
- IPv6 reachable OR: Parse any "a" lines, if present, and memorize whether at least one of them contains an IPv6 address. This indicates that at least one of the relay's IPv6 OR addresses is reachable.
- Relay flags: Parse relay flags from the "s" line. If there is no "Running" flag, skip this consensus entry. This ensures that we only consider running relays. Also parse any other relay flags from the "s" line that the relay had assigned.

If a consensus contains zero running relays, we skip it in the Relays by IP version (/relays-ipv6.html) graph, but not in the other graphs (simply because we didn't get around to changing those graphs). This is mostly to rule out a rare edge case when only a minority of directory authorities voted on the "Running" flag. In those cases, such a consensus would skew the average, even though relays were likely running.

Example

Consensus from 2018-03-14 12:00:00 with sample entry:

```
valid-after 2018-03-14 12:00:00
r BN00 APbV9tDVyUeuOUgZpQeBrsKF04U RgZNjWzBTrpkhlSMNyXHa6UIoo0 2018-03-13 19:17:49 51.
a [2001:bc8:4700:2300::23:315]:443
s Exit Fast Guard HSDir Running Stable V2Dir Valid
v Tor 0.3.3.3-alpha
pr Cons=1-2 Desc=1-2 DirCache=1-2 HSDir=1-2 HSIntro=3-4 HSRend=1-2 Link=1-5 LinkAuth=1,
w Bandwidth=4810
p reject 22,25,109-110,119,143,563,4899,6881-6889
```

Parsed sample consensus entry:

Description	Content
Server descriptor digest (converted to hex)	46064D8D6CC14EBA6486548C3725C76BA508A28D
IPv6 reachable OR	Yes
Relay flags	Exit, Fast, Guard, HSDir, Running, Stable, V2Dir, Valid

Step 2: Parse relay server descriptors

Parsing relay server descriptors is an optional step. You only need to do this if you want to break down the number of running relays by something that relays report in their server descriptors. This includes, among other things, the relay's platform string containing tor software version and operating system and whether the relay announced an IPv6 OR address or permitted exiting to IPv6 targets.

Obtain relay server descriptors from CollecTor (/collector.html#type-server-descriptor). Again, refer to the Tor directory protocol, version 3 (https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt) for details on the descriptor format.

Parse any or all of the following parts from each server descriptor:

- Tor version: Parse the tor software version from the "platform" line and memorize the first three dotted numbers from it. If the "platform" line does not begin with "Tor" followed by a space character and a dotted version number, memorize the version as "Other". If the platform line is missing, we skip this descriptor, which later leads to not counting this relay at all rather than including it in the "Other" group, which is slightly wrong. Note that consensus entries also contain a "v" line with the tor software version from the referenced descriptor, which we do not use, because it was not present in very old consensuses, but which should work just as well for recent consensus.
- Operating system: Parse the "platform" line and memorize whether it contains either one of the substrings "Linux", "Darwin" (macOS), "BSD", or "Windows". If the "platform" line contains neither of these substrings,

memorize the platform as "Other". If the platform line is missing, we skip this descriptor, which later leads to not counting this relay at all rather than including it in the "Other" group, which is slightly wrong.
- IPv6 announced OR: Parse any `"or-address"` lines and memorize whether at least one of them contains an IPv6 address. This indicates that the relay announced an IPv6 address.
- IPv6 exiting: Parse the `"ipv6-policy"` line, if present, and memorize whether it's different from "reject 1-65535". This indicates whether the relay permitted exiting to IPv6 targets. If the line is not present, memorize that the relay does not permit exiting to IPv6 targets.
- Server descriptor digest: Compute the SHA-1 digest, or determine it from the file name in case of archived descriptor tarballs.

Example

Server descriptor that is referenced from the consensus entry in the previous example (excerpt):

```
router BN00 51.15.56.204 443 0 80
or-address [2001:bc8:4700:2300::23:315]:443
platform Tor 0.3.3.3-alpha on Linux
published 2018-03-13 19:17:49
fingerprint 00F6 D5F6 D0D5 C947 AE39 4819 A507 81AE C285 D385
ipv6-policy reject 22,25,109-110,119,143,563,4899,6881-6889
```

Parsed server descriptor contents:

Description	Content
Tor version	0.3.3
Operating system	Linux
IPv6 announced OR	Yes
IPv6 exiting	Yes
Server descriptor digest	46064D8D6CC14EBA6486548C3725C76BA508A28D

Step 3: Compute daily averages

Optionally, match consensus entries with server descriptors by SHA-1 digest. Every consensus entry references exactly one server descriptor, and a server descriptor may be referenced from an arbitrary number of consensus entries. We handle missing server descriptors differently in the graphs covered in this section:

- Relays by tor version (/versions.html) and Relays by platform (/platforms.html): If a referenced server descriptor is missing, we also skip the consensus entry. We are aware that this is slightly wrong, because we should either exclude a consensus with too few matching server descriptors from the overall result, or at least count these relays as unknown tor version or unknown platform.
- Relays by IP version (/relays-ipv6.html): If at least 0.1% of referenced server descriptors are missing, we skip the consensus. We chose this threshold as low, because missing server descriptors may easily skew the results. However, a small number of missing server descriptors per consensus is acceptable and also unavoidable.

Go through all previously processed consensuses by valid-after UTC date. Compute the arithmetic mean of running relays, possibly broken down by relay flag, tor version, platform, or IPv6 capabilities, as the sum of all running relays divided by the number of consensuses. Round down to the next integer number.

Skip the last day of the results if it matches the current UTC date, because those averages may still change throughout the day. For the Relays by IP version (/relays-ipv6.html) graph we further skip days for which fewer than 12 consensuses are known. The goal is to avoid over-representing a few consensuses during periods when the directory authorities had trouble producing a consensus for at least half of the day.

Running bridges

After explaining our running relays statistics we continue with our running bridges statistics. The steps are quite similar, except for a couple differences in data formats that justify explaining these statistics in a separate subsection.

The following description applies to the following graphs:

- Relays and bridges (just the bridges part; for the relays part see above) graph (/networksize.html)
- Bridges by IP version graph (/bridges-ipv6.html)

Step 1: Parse bridge network statuses

Obtain bridge network statuses from CollecTor (/collector.html#type-bridge-network-status). Refer to the Tor bridge descriptors page (/bridge-descriptors.html) for details on the descriptor format.

Parse the bridge authority identify from the file name and memorize it. This is only relevant for times when more than 1 bridge authority was running. In those cases, bridges typically register at a single bridge authority only, so that taking the average of running bridges over all statuses on those day would be misleading.

Parse and memorize the `"published"` time either from the file name or from the status header. This timestamp is used to uniquely identify the status while processing, and the UTC date of this timestamp is later used to aggregate by UTC date.

Repeat the following steps for each status entry:

- Server descriptor digest: Parse the server descriptor digest from the `"r"` line and memorize it.
- Relay flags: Parse relay flags from the `"s"` line. If there is no `"Running"` flag, skip this entry. This ensures that we only consider running bridges.

If a status contains zero running bridges, skip it. This may happen when there is a temporary issue with the bridge authority.

Example

Bridge network status from 2018-03-14 12:04:07 with sample entry:

```
published 2018-03-14 12:04:07
r Lisbeth 2cgFyVXLEk0YjA1E8nHpvlfeIQk eyDMauE4xc5TotkgJiGHhMKGB3c 2018-03-14 09:58:34 
s Fast Guard Stable V2Dir Valid
w Bandwidth=8506
p reject 1-65535
```

Parsed sample bridge network status entry:

Description	Content
Server descriptor digest (converted to hex)	7B20CC6AE138C5CE53A2D92026218784C2860777

Step 2: Parse bridge server descriptors.

Parsing bridge server descriptors is an optional step. You only need to do this if you want to break down the number of running bridges by something that bridges report in their server descriptors. This includes, among other things, whether the bridge announced an IPv6 OR address.

Obtain bridge server descriptors from CollecTor (/collector.html#type-bridge-server-descriptor). As above, refer to the Tor bridge descriptors page (/bridge-descriptors.html) for details on the descriptor format.

Parse the following parts from each server descriptor:

- IPv6 announced OR: Parse any `"or-address"` lines and memorize whether at least one of them contains an IPv6 address. This indicates that the bridge announced an IPv6 address.
- Server descriptor digest: Parse the SHA-1 digest from the `"router-digest"` line, or determine it from the file name in case of archived descriptor tarballs.

Example

Server descriptor that is referenced from the bridge status entry in the previous example (excerpt):

```
router Lisbeth 10.197.150.133 61489 0 0
or-address [fd9f:2e19:3bcf::2f:ad91]:61489
router-digest-sha256 FP+uN2yMdpCB2koU76l1hdIxxtJ0Q/PB33wT9DWC4d0
router-digest 7B20CC6AE138C5CE53A2D92026218784C2860777
```

Parsed server descriptor contents:

Description	Content
IPv6 announced OR	Yes
Server descriptor digest	7B20CC6AE138C5CE53A2D92026218784C2860777

Step 3: Compute daily averages

Optionally, match status entries with server descriptors by SHA-1 digest. Every status entry references exactly one server descriptor, and a server descriptor may be referenced from an arbitrary number of status entries. If at least 0.1% of referenced server descriptors are missing, we skip the status. We chose this threshold as low, because missing server descriptors may easily skew the results. However, a small number of missing server descriptors per status is acceptable and also unavoidable.

We compute averages differently in the graphs covered in this section:

- Relays and bridges (/networksize.html): For each bridge authority, compute the arithmetic mean of running bridges as the sum of all running bridges divided by the number of statuses; sum up averages for all bridge authorities per day and round down to the next integer number.
- Bridges by IP version (/bridges-ipv6.html): Compute the arithmetic mean of running bridges as the sum of all running bridges divided by the number of statuses and round down to the next integer number. We are aware that this approach does not correctly reflect that bridges typically register at a single bridge authority only.

Skip the last day of the results if it matches the current UTC date, because those averages may still change throughout the day. For the Bridges by IP version (/bridges-ipv6.html) graph we further skip days for which fewer than 12 statuses are known. The goal is to avoid over-representing a few statuses during periods when the bridge directory authority had trouble producing a status for at least half of the day.

Traffic

Our traffic statistics have in common that their metrics are based on user-generated traffic. This includes advertised and consumed bandwidth and connection usage statistics.

Advertised bandwidth

Advertised bandwidth is the volume of traffic, both incoming and outgoing, that a relay is willing to sustain, as configured by the operator and claimed to be observed from recent data transfers. Relays self-report their advertised bandwidth in their server descriptors which we evaluate together with consensuses.

The following description applies to the following graphs:

- Total relay bandwidth (just the advertised bandwidth part; for the consumed bandwidth part see below) graph (/bandwidth.html)
- Advertised and consumed bandwidth by relay flag (just the advertised bandwidth part; for the consumed bandwidth part see below) graph (/bandwidth-flags.html)
- Advertised bandwidth by IP version graph (/advbw-ipv6.html)

- Advertised bandwidth distribution graph (/advbwdist-perc.html)
- Advertised bandwidth of n-th fastest relays graph (/advbwdist-relay.html)

Step 1: Parse relay server descriptors

Obtain relay server descriptors from CollecTor (/collector.html#type-server-descriptor). Refer to the Tor directory protocol, version 3 (https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt) for details on the descriptor format.

Parse the following parts from each server descriptor:

- Advertised bandwidth: Parse the three values (or just two in very old descriptors) from the `"bandwidth"` line. These values stand for the average bandwidth, burst bandwidth, and observed bandwidth. The advertised bandwidth is the minimum of these values.
- Server descriptor digest: Compute the SHA-1 digest, or determine it from the file name in case of archived descriptor tarballs.

Example

Server descriptor that is referenced from the consensus entry in the previous example (excerpt):

```
router CalyxInstitute04 162.247.73.204 9001 0 9030
bandwidth 10240000 10752000 7545479
```

Parsed server descriptor contents:

Description	Content
Advertised bandwidth	7545479 = min(10240000, 10752000, 7545479)
Server descriptor digest	78BB9105692521C58B2F103575ED7516634C45D4

Step 2: Parse consensuses

Obtain consensuses from CollecTor (/collector.html#type-network-status-consensus-3). Refer to the Tor directory protocol, version 3 (https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt) for details on the descriptor format.

From each consensus, parse the `"valid-after"` time from the header section.

From each consensus entry, parse the base64-encoded server descriptor digest from the `"r"` line. We are going to use this digest to match the entry with the advertised bandwidth value from server descriptors later on.

Also parse the relay flags from the `"s"` line. If there is no `"Running"` flag, skip this entry. (Consensuses with consensus method 4, introduced in 2008, or later do not list non-running relays, so that checking relay flags in recent consensuses is mostly done as a precaution without actual effect on the parsed data.) Further parse the `"Guard"`, `"Exit"`, and `"BadExit"` relay flags from this line. We consider a relay with the `"Guard"` flag as guard and a relay with the `"Exit"` and without the `"BadExit"` flag as exit.

Example

Original consensus (excerpt):

```
valid-after 2018-03-14 12:00:00
r CalyxInstitute04 UBs9vyULCUoFyl28QkrUw9RnIaI eLuRBWklIcWLLxA1de11FmNMRdQ 2018-03-14 1
s Exit Fast Guard HSDir Running Stable V2Dir Valid
v Tor 0.3.2.8-rc
pr Cons=1-2 Desc=1-2 DirCache=1-2 HSDir=1-2 HSIntro=3-4 HSRend=1-2 Link=1-4 LinkAuth=1,
w Bandwidth=5660
p accept 20-23,43,53,79-81,88,110,143,194,220,389,443,464,531,543-544,554,563,636,706,7
```

Parsed consensus contents:

Description	Content
Valid-after time	2018-03-14 12:00:00
Server descriptor digest (converted to hex)	78BB9105692521C58B2F103575ED7516634C45D4
Running	Yes
Guard	Yes
Exit	Yes

Step 3: Compute daily averages

The first three graphs described here, namely Total relay bandwidth (/bandwidth.html), Advertised and consumed bandwidth by relay flag (/bandwidth-flags.html), and Advertised bandwidth by IP version (/advbw-ipv6.html), have in common that they show daily averages of advertised bandwidth.

In order to compute these averages, first match consensus entries with server descriptors by SHA-1 digest. Every consensus entry references exactly one server descriptor, and a server descriptor may be referenced from an arbitrary number of consensus entries. We handle missing server descriptors differently in the graphs covered in this section:

- Total relay bandwidth (/bandwidth.html) and Advertised and consumed bandwidth by relay flag (/bandwidth-flags.html): If a referenced server descriptor is missing, we also skip the consensus entry. We are aware that this is slightly wrong, because we should rather exclude a consensus with too few matching server descriptors from the overall result than including it with an advertised bandwidth sum that is too low.
- Advertised bandwidth by IP version (/advbw-ipv6.html): If at least 0.1% of referenced server descriptors are missing, we skip the consensus. We chose this threshold as low, because missing server descriptors may easily skew the results. However, a small number of missing server descriptors per consensus is acceptable and also unavoidable.

Go through all previously processed consensuses by valid-after UTC date. Compute the arithmetic mean of advertised bandwidth as the sum of all advertised bandwidth values divided by the number of consensuses. Round down to the next integer number.

Break down numbers by guards and/or exits by taking into account which relay flags a consensus entry had that referenced a server descriptor.

Skip the last day of the results if it matches the current UTC date, because those averages may still change throughout the day. For the Advertised bandwidth by IP version (/advbw-ipv6.html) graph we further skip days for which fewer than 12 consensuses are known. The goal is to avoid over-representing a few consensuses during periods when the directory authorities had trouble producing a consensus for at least half of the day.

Step 4: Compute ranks and percentiles

The remaining two graphs described here, namely Advertised bandwidth distribution (/advbwdist-perc.html) and Advertised bandwidth of n-th fastest relays (/advbwdist-relay.html), display advertised bandwidth ranks or percentiles.

Similar to the previous step, match consensus entries with server descriptors by SHA-1 digest. We handle missing server descriptors by simply skipping the consensus entry, at the risk of over-representing available server descriptors in consensuses where most server descriptors are missing.

For the Advertised bandwidth distribution (/advbwdist-perc.html) graph, determine the i-th percentile value for each consensus. We use a non-standard percentile definition that is loosely based on the nearest-rank method: the P-th percentile (`0 ≤ P ≤ 100`) of a list of `N` ordered values (sorted from least to greatest) is the *largest* value in the list such that no more than `P` percent of the data is strictly less than the value and at least `P` percent of the data is less than or equal to that value. We calculate the ordinal rank `n` using the following formula: `floor((P / 100) * (N - 1)) + 1`

Calculate the median value over all consensus from a given day for each percentile value.

Consider the set of all running relays as well as the set of exit relays.

For the Advertised bandwidth of n-th fastest relays (/advbwdist-relay.html) graph, determine the n-th highest advertised bandwidth value for each consensus, and then calculate the median value over all consensus from a given day. Again consider the set of all running relays as well as the set of exit relays.

Consumed bandwidth

Consumed bandwidth, or bandwidth history, is the volume of incoming and/or outgoing traffic that a relay claims to have handled on behalf of clients. Relays self-report bandwidth histories as part of their extra-info descriptors, which we evaluate in combination with consensuses.

The following description applies to the following graphs:

- Total relay bandwidth (just the consumed bandwidth part; for the advertised bandwidth part see above) graph (/bandwidth.html)
- Advertised and consumed bandwidth by relay flag (just the consumed bandwidth part; for the advertised bandwidth part see above) graph (/bandwidth-flags.html)
- Consumed bandwidth by Exit/Guard flag combination graph (/bwhist-flags.html)
- Bandwidth spent on answering directory requests graph (/dirbytes.html)

Step 1: Parse extra-info descriptors

Obtain extra-info descriptors from CollecTor (/collector.html#type-extra-info). Refer to the Tor directory protocol, version 3 (https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt) for details on the descriptor format.

Parse the fingerprint from the `"extra-info"` line. We will use this fingerprint to deduplicate statistics included in other extra-info descriptor published by the same relay. We may also use this fingerprint to attribute statistics to relays with the `"Exit"` and/or `"Guard"` flag.

Parse the `"write-history"`, `"read-history"`, `"dirreq-write-history"`, and `"dirreq-read-history` lines containing consumed bandwidth statistics. The first two histories include all bytes written or read by the relay, whereas the last two include only bytes spent on answering directory requests. If a statistics interval spans more than 1 UTC date, split observations to the covered UTC dates by assuming a linear distribution of observations. As a simplification, we shift reported statistics intervals forward to fully align with multiples of 15 minutes since midnight. We also discard reported statistics with intervals that are not multiples of 15 minutes.

Example

Original extra-info descriptor:

```
extra-info CalyxInstitute04 501B3DBF250B094A05CA5DBC424AD4C3D46721A2
write-history 2018-03-14 03:32:21 (86400 s) 332146548736,317531642880,226877641728,274
read-history 2018-03-14 03:32:21 (86400 s) 336636388352,317718054912,224448458752,2752
dirreq-write-history 2018-03-14 03:32:21 (86400 s) 28716032,73700352,7670784,18254848,
dirreq-read-history 2018-03-14 03:32:21 (86400 s) 6373376,8518656,9504768,13232128,118
```

Parsed extra-info descriptor contents (subset):

Description	Content
Fingerprint	501B3DBF250B094A05CA5DBC424AD4C3D46721A2
Written bytes	261 GiB = 81 * floor(332146548736 B / 96) between 2018-03-09 03:45:00 and 2018-03-10 00:00:00
	48 GiB = 15 * floor(332146548736 B / 96) between 2018-03-10 00:00:00 and 2018-03-10 03:45:00
Read bytes	265 GiB = 81 * floor(336636388352 B / 96) between 2018-03-09 03:45:00 and 2018-03-10 00:00:00

	49 GiB = 15 * floor(336636388352 B / 96) between 2018-03-10 00:00:00 and 2018-03-10 03:45:00
Written directory bytes	23 MiB = 81 * floor(28716032 B / 96) between 2018-03-09 03:45:00 and 2018-03-10 00:00:00
	4 MiB = 15 * floor(28716032 B / 96) between 2018-03-10 00:00:00 and 2018-03-10 03:45:00
Read directory bytes	5 MiB = 81 * floor(6373376 B / 96) between 2018-03-09 03:45:00 and 2018-03-10 00:00:00
	1 MiB = 15 * floor(6373376 B / 96) between 2018-03-10 00:00:00 and 2018-03-10 03:45:00

Step 2: Parse consensuses

Obtain consensuses from CollecTor (/collector.html#type-network-status-consensus-3). Refer to the Tor directory protocol, version 3 (https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt) for details on the descriptor format.

From each consensus, parse the `"valid-after"` time from the header section.

From each consensus entry, parse the base64-encoded relay fingerprint from the `"r"` line. We are going to use this fingerprint to match the entry with statistics from extra-info descriptors later on.

Also parse the relay flags from the `"s"` line. If there is no `"Running"` flag, skip this entry. (Consensuses with consensus method 4, introduced in 2008, or later do not list non-running relays, so that checking relay flags in recent consensuses is mostly done as a precaution without actual effect on the parsed data.) Further parse the `"Guard"`, `"Exit"`, and `"BadExit"` relay flags from this line. We consider a relay with the `"Guard"` flag as guard and a relay with the `"Exit"` and without the `"BadExit"` flag as exit.

Example

Original consensus (excerpt):

```
valid-after 2018-03-14 12:00:00
r CalyxInstitute04 UBs9vyULCUoFyl28QkrUw9RnIaI eLuRBWklIcWLLxA1de11FmNMRdQ 2018-03-14 1
s Exit Fast Guard HSDir Running Stable V2Dir Valid
v Tor 0.3.2.8-rc
pr Cons=1-2 Desc=1-2 DirCache=1-2 HSDir=1-2 HSIntro=3-4 HSRend=1-2 Link=1-4 LinkAuth=1,
w Bandwidth=5660
p accept 20-23,43,53,79-81,88,110,143,194,220,389,443,464,531,543-544,554,563,636,706,7
```

Parsed consensus contents:

Description	Content
Valid-after time	2018-03-14 12:00:00
Fingerprint (converted to hex)	501B3DBF250B094A05CA5DBC424AD4C3D46721A2
Running	Yes
Guard	Yes
Exit	Yes

Step 3: Compute daily totals

The first three graphs described here, namely Total relay bandwidth (/bandwidth.html), Advertised and consumed bandwidth by relay flag (/bandwidth-flags.html), and Consumed bandwidth by Exit/Guard flag combination (/bwhist-flags.html), show daily totals of all bytes written or read by relays.

The total consumed bandwidth for the first graph, Total relay bandwidth (/bandwidth.html), is the easiest to compute: sum up all read and written bytes on a given day and divide the result by 2.

The second and third graph, Advertised and consumed bandwidth by relay flag (/bandwidth-flags.html) and Consumed bandwidth by Exit/Guard flag combination (/bwhist-flags.html), require us to match consensus entries with extra-info descriptors by relay fingerprint. We only include bandwidth histories if a relay was listed as running at least once on a day, and we attribute bandwidth to guards and/or exits if a relay was a guard and/or exit at least in one consensus on a day.

We're currently discussing changing the first graph (https://bugs.torproject.org/26015) by only including bandwidth histories of relays that were listed as running at least once on a day, too.

The fourth graph, Bandwidth spent on answering directory requests (/dirbytes.html), right now shows an estimate of bytes spent by directory mirrors on answering directory requests. We're currently working on simplifying that graph (https://bugs.torproject.org/26002) and will add a description for aggregating these values after making those changes.

Connection usage

The last category of traffic statistics concerns statistics on the fraction of connections used uni- or bidirectionally. A subset of relays reports these highly aggregated statistics in their extra-info descriptors.

The following description applies to the following graph:

- Fraction of connections used uni-/bidirectionally graph (/connbidirect.html)

Step 1: Parse relay extra-info descriptors

Obtain relay extra-info descriptors from CollecTor (/collector.html#type-extra-info).

Parse the relay fingerprint from the `"extra-info"` line. We deduplicate reported statistics by UTC date of reported statistics and relay fingerprint. If a given relay publishes different statistics on a given UTC day, we pick the first encountered statistics and discard all subsequent statistics by that relay on that UTC day.

Parse the UTC date from the `"conn-bi-direct"` line. We use this date to aggregate statistics, regardless of what period of time fell on the statistics end date as opposed to the previous date.

From the same line, parse the three counts `READ`, `WRITE`, and `BOTH`, but disregard the `BELOW` value. Discard any statistics where the sum of these three values is 0. Compute three fractions by dividing each of the three values `READ`, `WRITE`, and `BOTH` by the sum of all three. Multiply results with 100 and truncate any decimal places, keeping a fraction value between 0 and 100.

Example

Original extra-info descriptor:

```
extra-info CalyxInstitute04 501B3DBF250B094A05CA5DBC424AD4C3D46721A2
conn-bi-direct 2018-03-14 02:50:57 (86400 s) 2761409,10595,231429,289346
```

Parsed extra-info descriptor contents:

Description	Content
Fingerprint	501B3DBF250B094A05CA5DBC424AD4C3D46721A2
Date	2018-03-14
Mostly reading	1 = floor((10595 / (10595 + 231429 + 289346)) * 100)
Mostly writing	43 = floor((231429 / (10595 + 231429 + 289346)) * 100)
Both reading and writing	54 = floor((289346 / (10595 + 231429 + 289346)) * 100)

Step 2: Aggregate statistics

For each date, compute the 25th, 50th, and 75th percentile of fractions computed in the previous step.

We use a non-standard percentile definition that is similar to the nearest-rank method: the P-th percentile (`0 < P ≤ 100`) of a list of `N` ordered values (sorted from least to greatest) is the *largest* value in the list such that no more than `P` percent of the data is strictly less than the value and at least `P` percent of the data is less than or equal to that value. We calculate the ordinal rank `n` using the following formula: `floor((P / 100) * N) + 1`

Example

Parsed and aggregated statistics for 2018-03-14:

Description	Content
Mostly reading fractions with 25th/50th/75th percentiles highlighted	0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, **5**, 6, 9, 10, 12, 14, 14, 15, 15, 15, 16, 16, 16, 17, 17, 17, 17, 17, 17, 17, 18, 18, 18, 18, **19**, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, **21**, 21, 21, 21, 21, 22, 22, 22, 22, 22, 22, 22, 23, 23, 23, 23, 23, 24, 24, 25, 25, 26, 33, 40, 43
Mostly writing fractions with 25th/50th/75th percentiles highlighted	8, 13, 13, 14, 15, 16, 16, 16, 16, 16, 16, 17, 17, 17, 17, 17, 17, 18, 18, 18, 18, 18, 18, 18, **18**, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 20, 20, 20, 20, 20, 20, 20, 20, 20, **20**, 20, 21, 21, 21, 21, 22, 22, 22, 22, 23, 23, 24, 26, 26, 31, 33, 38, 38, 39, 39, 40, 40, 40, **40**, 40, 41, 41, 43, 43, 43, 43, 43, 43, 44, 44, 44, 44, 44, 45, 45, 45, 46, 46, 49, 50, 56, 61, 95
Both reading and writing fractions with 25th/50th/75th percentiles highlighted	4, 10, 13, 29, 30, 33, 36, 41, 42, 44, 45, 47, 51, 51, 52, 52, 52, 52, 52, 53, 53, 53, 54, 54, **54**, 54, 54, 54, 54, 54, 54, 55, 55, 55, 55, 55, 55, 55, 56, 56, 56, 56, 56, 56, 57, 57, 57, 57, **57**, 57, 57, 57, 58, 58, 58, 58, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 60, 60, 60, 60, 60, 61, **61**, 61, 61, 61, 62, 62, 62, 63, 63, 63, 64, 64, 64, 65, 65, 65, 65, 65, 66, 66, 68, 69, 70, 73, 80
Number of values	97
Ordinal ranks	`25 = floor((25 / 100) * 97) + 1`
	`49 = floor((50 / 100) * 97) + 1`
	`73 = floor((75 / 100) * 97) + 1`
Mostly reading fraction, 25th/50th/75th percentiles	5; 19; 21
Mostly writing fraction, 25th/50th/75th percentiles	18; 20; 40
Both reading and writing fraction, 25th/50th/75th percentiles	54; 57; 61

Performance

We perform active measurements of Tor network performance by running several OnionPerf (https://github.com/robgjansen/onionperf) (previously: Torperf (https://gitweb.torproject.org/torperf.git)) instances from different vantage points. Here we explain how we evaluate Torperf/OnionPerf measurement to obtain the same results as on Tor Metrics.

The following description applies to the following graphs:

- Time to download files over Tor graph (/torperf.html)

- Timeouts and failures of downloading files over Tor graph (/torperf-failures.html)

Step 1: Parse OnionPerf and/or Torperf measurement results

Obtain OnionPerf/Torperf measurement results from CollecTor (/collector.html#type-torperf).

From each measurement result, parse at least the following keys:

- `SOURCE` : Configured name of the data source.
- `FILESIZE` : Configured file size in bytes.
- `START` : Download start time that we use for two purposes: to determine how long a request took and to aggregate measurements by date.
- `DATACOMPLETE` : Download end time that is only set if the request succeeded.
- `ENDPOINTREMOTE` : Hostname, IP address, and port that was used to connect to the remote server; we use this to distinguish a request to a public server (if `ENDPOINTREMOTE` is not present or does not contain `".onion"` as substring) or to an onion server.
- `DIDTIMEOUT` : 1 if the request timed out, 0 otherwise.
- `READBYTES` : Total number of bytes read, which indicates whether this request succeeded (if ≥ `FILESIZE`) or failed.

Example

Original OnionPerf measurement result (with line breaks and indentations added for clarity only):

```
BUILDTIMES=0.0599999427795,0.230000019073,0.5
CIRC_ID=63777
CONNECT=1521009192.60
DATACOMPLETE=1521009204.70
DATAPERC0=1521009193.20
DATAPERC10=1521009196.34
DATAPERC100=1521009204.70
DATAPERC20=1521009197.80
DATAPERC30=1521009198.92
DATAPERC40=1521009199.95
DATAPERC50=1521009200.72
DATAPERC60=1521009201.50
DATAPERC70=1521009202.33
DATAPERC80=1521009203.09
DATAPERC90=1521009203.91
DATAREQUEST=1521009192.90
DATARESPONSE=1521009193.20
DIDTIMEOUT=0
ENDPOINTLOCAL=localhost:127.0.0.1:60970
ENDPOINTPROXY=localhost:127.0.0.1:25855
ENDPOINTREMOTE=37.218.247.40:37.218.247.40:80
FILESIZE=5242880
HOSTNAMELOCAL=op-nl
HOSTNAMEREMOTE=op-nl
LAUNCH=1521008952.17
NEGOTIATE=1521009192.60
PATH=$21D53193494ADB42057242CE276953A90543BE31,
  $D31BFE2048AD5B77CA072E977B5961861A2415E6,
  $FE9733F8403BA4215222B9C4CBE9386EE849732C
QUANTILE=0.8
READBYTES=5242950
REQUEST=1521009192.61
RESPONSE=1521009192.90
SOCKET=1521009192.60
SOURCE=op-nl
SOURCEADDRESS=37.218.247.40
START=1521009192.60
TIMEOUT=1500
USED_AT=1521009204.7
USED_BY=106998
WRITEBYTES=56
```

Parsed OnionPerf measurement result:

Description	Content
Source name	op-nl
Requested static file size	5242880 bytes
Measurement start	2018-03-14 06:33:12.600
Measurement end	2018-03-14 06:33:24.700
Server type	Public server (37.218.247.40 is not an onion address)
Request timeout	No
Read bytes	5242950 (≥ 5242880)

Step 2: Aggregate measurement results

Each of the measurement results parsed in the previous steps constitutes a single measurement. Ideally, most of these measurements would succeed, but it's also possible that some measurements did not complete within a pre-defined timeout or failed for some other reason. We distinguish three cases:

- Timeouts: measurements that either timed out (with `DIDTIMEOUT = 1`) or that have an invalid measurement end time (`DATACOMPLETE ≤ START`),
- Failures: measurements that did not time out (with `DIDTIMEOUT = 0`), that had a valid measurement end time (`DATACOMPLETE > START`), and that had fewer bytes read than expected (`READBYTES < FILESIZE`), and
- Requests: all measurements, including successes, timeouts, and failures.

In addition to request counts, we are also interested in statistics on download times. Therefore we consider only measurements with `DATACOMPLETE > START` , for which we calculate the download time as: `DATACOMPLETE - START` . We then compute the 25th, 50th, and 75th percentile of download times by sorting download times, determining the percentile rank, and using linear interpolation between adjacent ranks.

Example

Measurements from OnionPerf instance op-nl downloading static files with a size of 5242880 bytes from its public server on 2018-03-14, sorted by download time:

Request start	Request end	Request duration in seconds	Timeout	Bytes read
02:38:12.570	02:38:14.880	2.310	No	5242950
03:18:12.580	03:18:15.660	3.080	No	5242950
03:03:12.580	03:03:16.060	3.480	No	5242950
01:18:12.570	01:18:16.660	4.090	No	5242950
22:38:12.700	22:38:16.800	4.100	No	5242950
00:33:12.560	00:33:18.860	6.300	No	5242950
05:43:12.590	05:43:19.120	6.530	No	5242950
17:28:12.670	17:28:19.390	6.720	No	5242950
11:13:12.630	11:13:20.290	7.660	No	5242950
19:23:12.680	19:23:20.510	7.830	No	5242950
16:58:12.670	16:58:21.220	8.550	No	5242950
06:33:12.600	06:33:24.700	12.100	No	5242950

Aggregates for OnionPerf instance op-nl downloading static files with a size of 5242880 bytes from its public server on 2018-03-14:

Description	Content
Timeout count	0
Failure count	0
Success count	12
25th percentile	3.9375 = 3.480 + 0.75 * (4.090 - 3.480) with rank being 3.75 = 25 / 100 * (12 - 1) + 1
50th percentile	6.415 = 6.300 + 0.5 * (6.530 - 6.300) with rank being 6.5 = 50 / 100 * (12 - 1) + 1
75th percentile	7.7025 = 7.660 + 0.25 * (7.830 - 7.660) with rank being 9.25 = 75 / 100 * (12 - 1) + 1

Onion Services

Our onion services statistics are based on two statistics reported by relays that have been added in 2014 to give some first insights into onion-service usage. For further background on the following steps, refer to the technical report titled "Extrapolating network totals from hidden-service statistics" (https://research.torproject.org/techreports/extrapolating-hidserv-stats-2015-01-31.pdf) that this description is based on (which was written before hidden services were renamed to onion services).

The following description applies to the following graphs:

- Unique .onion addresses (version 2 only) graph (/hidserv-dir-onions-seen.html)
- Onion-service traffic (versions 2 and 3) graph (/hidserv-rend-relayed-cells.html)
- Fraction of relays reporting onion-service statistics graph (/hidserv-frac-reporting.html)

Step 1: Parse reported statistics from extra-info descriptors

Obtain relay extra-info descriptors from CollecTor (/collector.html#type-extra-info).

Parse the following parts from each extra-info descriptor:

- Relay fingerprint: The `"extra-info"` line tells us which relay reported these statistics, which we need to know to match them with the expected fraction of onion-service activity throughout the statistics interval.
- Onion service statistics interval end: The `"hidserv-stats-end"` line tells us when the statistics interval ended, and, together with the interval length, when it started.
- Cells relayed as rendezvous point: The `"hidserv-rend-relayed-cells"` line tells us the number of cells that the relay handled on rendezvous circuits, and it tells us how this number has been obfuscated by the relay. The value for `"bin_size"` is the bin size used for rounding up the originally observed cell number, and the values for `"delta_f"` and `"epsilon"` are inputs for the additive noise following a Laplace distribution.
- .onion addresses observed as directory: Finally, the `"hidserv-dir-onions-seen"` line tells us the number of .onion addresses that the relay observed in published onion-service descriptors in its role as onion-service directory.

Note: Unlike other statistics, we're not splitting statistics by UTC date. Instead, we're only accepting statistics intervals that are exactly 1 day long, and we're counting all reported values for the UTC date of the statistics end time.

Example

Sample onion-service statistics contained in an extra-info descriptor:

```
extra-info EffSSLObservatory6 08EBE0BD789D2C51E9A148FA486EDAD8DA0A4713
[...]
hidserv-stats-end 2018-03-14 01:23:35 (86400 s)
hidserv-rend-relayed-cells 1368667 delta_f=2048 epsilon=0.30 bin_size=1024
hidserv-dir-onions-seen 353 delta_f=8 epsilon=0.30 bin_size=8
```

Parsed extra-info descriptor contents:

Description	Content
Fingerprint	08EBE0BD789D2C51E9A148FA486EDAD8DA0A4713
Onion service statistics interval end	2018-03-14 01:23:35
Cells relayed as rendezvous point	1368667 with obfuscation parameters: delta_f=2048 epsilon=0.30 bin_size=1024
.onion addresses observed as directory	353 with obfuscation parameters: delta_f=8 epsilon=0.30 bin_size=8

Step 2: Remove previously added noise

When processing onion-service statistics, we need to handle the fact that they have been obfuscated by relays. As first step, we're attempting to remove the additive Laplace-distributed noise by rounding up to the nearest multiple of `bin_size`. The idea is that it's most likely that noise was added to the closest right side of a bin than to the right side of another bin. In step two, we're subtracting half of `bin_size`, because the relay added between 0 and `bin_size − 1` to the originally observed value. All in all, we're using the following formula to remove previously added noise:

```
floor((reported + binSize / 2) / binSize) * binSize - binSize / 2
```

Note: Our Java implementation of this formula uses `trunc()` rather than `floor()` and hence provides different results for negative reported statistics; this is currently under discussion (https://bugs.torproject.org/26022).

Example

Sample statistics from above after removing noise:

Cells relayed as rendezvous point	1368576 = floor((1368667 + 512) / 1024) * 1024 - 512
.onion addresses observed as directory	348 = floor((353 + 4) / 8) * 8 - 4

Step 3: Parse consensuses

Obtain consensuses from CollecTor (/collector.html#type-network-status-consensus-3).

From each consensus, parse the `"valid-after"` time from the header section.

From each consensus entry, parse the base64-encoded relay fingerprint from the `"r"` line.

Also parse the relay flags from the `"s"` line. If there is no `"Running"` flag, skip this entry. (Consensuses with consensus method 4, introduced in 2008, or later do not list non-running relays, so that checking relay flags in recent consensuses is mostly done as a precaution without actual effect on the parsed data.) Parse the remaining relay flags from this line.

Finally, parse the weights contained in the `"bandwidth-weights"` line from the footer section fo the consensus.

Step 4: Derive network fractions from consensuses

The probability of choosing a relay as rendezvous point varies a lot between relays, and not all onion-service directories handle the same number of onion-service descriptors. Fortunately, we can derive what fraction of rendezvous circuits a relay has handled and what fraction of descriptors a directory was responsible for.

The first fraction that we compute is the probability of a relay to be selected as rendezvous point. Clients only select relays as rendezvous point that have the `"Fast"` flag. They weight relays differently based on their bandwidth and depending on whether they have the `"Exit"` and/or `"Guard"` flags: they weight the bandwidth value contained in the `"w"` line with the value of `"Wmg"`, `"Wme"`, `"Wmd"`, or `"Wmm"`, depending on whether the relay has only the `"Guard"` flag, only the `"Exit"` flag, both such flags, or neither of them.

The second fraction that we can derive from this consensus entry is the fraction of descriptor space that this relay was responsible for in its role as onion-service directory. The Tor Rendezvous Specification (https://gitweb.torproject.org/torspec.git/tree/rend-spec-v2.txt) contains the following definition: *"A[n onion] service directory is deemed responsible for a descriptor ID if it has the HSDir flag and its identity digest is one of the first three identity digests of HSDir relays following the descriptor ID in a circular list."*

Based on the fraction of descriptor space that a directory was responsible for we can compute the fraction of descriptors that this directory has seen. Intuitively, one might think that these fractions are the same. However, this is not the case: each descriptor that is published to a directory is also published to two other directories. As a result we need to divide the fraction of descriptor space by *three* to obtain the fraction of descriptors observed the directory. Note that, without dividing by three, fractions of all directories would not add up to 100 %.

Note: Not sure how to describe this: We calculate network fraction per consensus. When we extrapolate reported statistics, we compute the average (arithmetic mean) of all such network fractions with consensus valid-after times falling into the statistics interval. In particular, we're *not* computing the average of network fractions from the UTC day when the statistics interval ends; even though we're attributing extrapolated statistics to the UTC date of the statistics interval end in the next step.

Example

Sample consensus entries of relay `EffSSLObservatory6` that reports onion-service statistics and of the three onion-service directories preceding it:

```
valid-after 2018-03-14 12:00:00
[...]
r goatrelay CKq3DxK31fsFeGOnMGE8+M+7N8g gn3OhGn1S4w73+NbCVJOluUbsbk 2018-03-14 08:36:0
6 178.32.222.125 9001 0
s Fast Guard HSDir Running Stable V2Dir Valid
[...]
r PyRoTOR CNAdBMbzxsb0F9PdHuObVEw8vQo 2nfOeR7qREil1oCPMevVY99Int0 2018-03-14 02:00:34
192.210.192.229 9001 0
s Fast HSDir Running Stable V2Dir Valid
[...]
r ivitor COlTCzmfenGTYGZYazGmaHcGZFg ENDCOR1+c7Nr2/ENen+Vq3FPTsY 2018-03-14 04:43:54 8
s Fast HSDir Running Stable V2Dir Valid
[...]
r EffSSLObservatory6 COvgvXidLFHpoUj6SG7a2NoKRxM RAbwSXck4UwrripVf6fZRHLrUQc 2018-03-1
s Fast HSDir Running Stable V2Dir Valid
v Tor 0.3.1.9
pr Cons=1-2 Desc=1-2 DirCache=1-2 HSDir=1-2 HSIntro=3-4 HSRend=1-2 Link=1-4 LinkAuth=1,
w Bandwidth=798
p reject 1-65535
[...]
bandwidth-weights Wbd=0 Wbe=0 Wbg=3031 Wbm=10000 Wdb=10000 Web=10000 Wed=10000 Wee=100
```

Sample network fractions:

- Our sample relay, `EffSSLObservatory6`, has the `"Fast"` flag, a bandwidth value of 798, and neither `"Guard"` nor `"Exit"` flag. Its probability for being selected as rendezvous point is calculated as 798 × 10000/10000 = 798 divided by the sum of all such weights in the consensus. The result is 0.0047 %.
- In the sample consensus entry, we'd extract the base64-encoded fingerprint of the statistics-reporting relay, `UBs9vyU...`, and the fingerprint of the onion-service directory that precedes the relay by three positions, `T/jtfAc...`, and compute what fraction of descriptor space that is, in this case 0.0994 %. So, the relay has observed 0.0331 % of descriptors in the network.

Step 5: Extrapolate network totals

We are now ready to extrapolate network totals from reported statistics. We do this by dividing reported statistics by the calculated fraction of observations made by the reporting relay. The underlying assumption is that statistics grow linearly with calculated fractions. We only exclude relays from this step that have a calculated fraction of exactly zero, to avoid dividing by zero.

While we can expect this method to work as described for extrapolating cells on rendezvous circuits, we need to take another step for estimating the number of unique .onion addresses in the network. The reason is that a .onion address is not only known to a single relay, but to a couple of relays, all of which include that .onion address in their statistics. We need to subtract out the multiple counting of .onion addresses to come up with a network-wide number of unique .onion addresses.

As an approximation, we assume that an onion service publishes its descriptor to *twelve* directories over a 24-hour period: the service stores *two* replicas per descriptor using different descriptor identifiers, both descriptor replicas get stored to *three* different onion-service directories each, and the service changes descriptor identifiers once every 24 hours which leads to *two* different descriptor identifiers per replica.

To be clear, this approximation is not entirely accurate. For example, the two replicas or the descriptors with changed descriptor identifiers could have been stored to the same directory. As another example, onion-service directories might have joined or left the network and other directories might have become responsible for storing a descriptor which also include that .onion address in their statistics. However, for the subsequent analysis, we assume that neither of these cases affects results substantially.

Example

Sample extrapolated network totals:

- Our sample relay, `EffSSLObservatory6`, reported 1368576 cells relayed as rendezvous point. We calculated its probability for being selected as rendezvous point as 0.0047 %. Note that this probability can change over the day, and we're using the arithmetic mean of all values for a given day. From these values we would expect the network total to be 1368576 / 0.0047 % = 29118638298 cells. Converted to bit/s we obtain 29118638298 cells/day × 512 bytes/cell × 8 bits/bytes / 86400 s/day = 1380439149 bit/s = 1.29 Gbit/s.
- The same relay reported to have observed 348 .onion addresses as directory. The calculated network fraction is 0.0418 %. Combined, we would expect there to be 348 / 0.0418 % / 12 = 69378 .onion addresses in the network.

Step 6: Compute daily averages

As last step in the analysis, we aggregate extrapolated network totals from all reporting relays to obtain a daily average. We're using the weighted interquartile mean as metric, because it is robust against noisy statistics and potentially lying relays and considers half of the reported statistics. For this metric we order extrapolated network totals by their value, discard the lower and the upper quartile by weight, and compute the weighted mean of the remaining values.

We further define a threshold of 1 % for the total fraction of relays reporting statistics. If less than these 1% of relays report statistics on a given day, we don't include that day in the end results.

Applications

Our applications statistics are based on Tor web server requests where our users download applications initially and where they ask for updates.

The following description applies to the following graphs:

- Tor Browser downloads and updates graph (/webstats-tb.html)
- Tor Browser downloads by platform graph (/webstats-tb-platform.html)
- Tor Browser downloads by locale graph (/webstats-tb-locale.html)
- Tor Messenger downloads and updates graph (/webstats-tm.html)

Step 1: Parse Tor web server logs

Obtain Tor web server logs from CollecTor (/collector.html#type-webstats). Refer to the separate specification page (/web-server-logs.html) for details on the data format.

Each log file contains relevant meta data in its file name, including the site name, the server name, and the log date. The log file itself contains sanitized requests to Tor web servers.

All patterns mentioned in the following are understood by PostgreSQL's `LIKE` operator. An underscore (`_`) matches any single character; a percent sign (`%`) matches any string of zero or more characters.

Step 2: Count Tor Browser initial downloads

We count a request as Tor Browser initial download if it matches the following criteria:

- Request method: GET
- Resource string: `'%/torbrowser/%.exe'`, `'%/torbrowser/%.dmg'`, or `'%/torbrowser/%.tar.xz'`
- Response code: 200

We distinguish platforms based on the resource string: `'%.exe%'` for Windows, `'%.dmg%'` for macOS, and

`'%.tar.xz%'` for Linux.

We distinguish release channels based on the resource string: `'%-hardened%'` for hardened releases, `'%/%.%a%/%'` for alpha releases, and stable releases otherwise.

We extract the locale (for example, `'en-US'` for English as used in the United States or `'de'` for German) from the resource string using regular expression `'.*_([a-zA-Z]{2}|[a-zA-Z]{2}-[a-zA-Z]{2})[\._-].*'`, falling back to `'??'` for unrecognized locales if the regular expression does not match.

Example

Request from sanitized Tor web server log for virtual host dist.torproject.org from 2018-03-14:

```
0.0.0.1 - - [14/Mar/2018:00:00:00 +0000] "GET /torbrowser/7.5.1/torbrowser-install-7.5.
```

Step 3: Count Tor Browser signature downloads

We count a request as Tor Browser signature download if it matches the following criteria:

- Request method: GET
- Resource string: `'%/torbrowser/%.exe.asc'`, `'%/torbrowser/%.dmg.asc'`, or `'%/torbrowser/%.tar.xz.asc'`
- Response code: 200

We break down requests by platform, channel, and locale in the exact same way as for Tor Browser initial downloads (see above).

Example

Request from sanitized Tor web server log for virtual host dist.torproject.org from 2018-03-14:

```
0.0.0.1 - - [14/Mar/2018:00:00:00 +0000] "GET /torbrowser/7.5.1/tor-browser-linux64-7.5
```

Step 4: Count Tor Browser update pings

We count a request as Tor Browser update ping if it matches the following criteria:

- Request method: GET
- Resource string: `'%/torbrowser/update\__/%'` but not `'%.xml'`
- Response code: 200

We distinguish platforms based on the resource string: `'%/WINNT%'` for Windows, `'%/Darwin%'` for macOS, and Linux otherwise.

We distinguish release channels based on the resource string: `'%/hardened/%'` for hardened releases, `'%/alpha/%'` for alpha releases, and `'%/release/%'` for stable releases.

We extract the locale (for example, `'en-US'` for English as used in the United States or `'de'` for German) from the resource string using regular expression `'.*/([a-zA-Z]{2}|[a-zA-Z]{2}-[a-zA-Z]{2})\??$'`, falling back to `'??'` for unrecognized locales if the regular expression does not match.

Example

Request from sanitized Tor web server log for virtual host aus1.torproject.org from 2018-03-14:

```
0.0.0.1 - - [14/Mar/2018:00:00:00 +0000] "GET /torbrowser/update_3/release/WINNT_x86-g
```

Step 5: Count Tor Browser update requests

We count a request as Tor Browser update request if it matches the following criteria:

- Request method: GET
- Resource string: `'%/torbrowser/%.mar'`
- Response code: 302

We distinguish platforms based on the resource string: `'%-win32-%'` for Windows, `'%-osx%'` for macOS, and Linux otherwise.

We distinguish release channels based on the resource string: `'%-hardened%'` for hardened releases, `'%/%.%a%/%'` for alpha releases, and stable releases otherwise.

We extract the locale (for example, `'en-US'` for English as used in the United States or `'de'` for German) from the resource string using regular expression `'.*_([a-zA-Z]{2}|[a-zA-Z]{2}-[a-zA-Z]{2})[\._-].*'`, falling back to `'??'` for unrecognized locales if the regular expression does not match.

We distinguish incremental updates having `'%.incremental.%'` in the resource string from non-incremental (full) updates that don't contain this pattern in their resource string.

Example

Request from sanitized Tor web server log for virtual host cdn.torproject.org from 2018-03-14:

```
0.0.0.1 - - [14/Mar/2018:00:00:00 +0000] "GET /aus1/torbrowser/7.5.1/tor-browser-win32-
```

Step 6: Count Tor Messenger initial downloads

We count a request as Tor Messenger initial download if it matches the following criteria:

- Request method: GET
- Resource string: `'%/tormessenger/%.exe'`, `'%/tormessenger/%.dmg'`, and `'%/tormessenger/%.tar.xz'`
- Response code: 200

We distinguish platforms based on the resource string: `'%.exe'` for Windows, `'%.dmg'` for macOS, and `'%.tar.xz'` for Linux.

We extract the locale (for example, `'en-US'` for English as used in the United States or `'de'` for German) from the resource string using regular expression `'.*_([a-zA-Z]{2}|[a-zA-Z]{2}-[a-zA-Z]{2})[\._-].*'`, falling back to `'??'` for unrecognized locales if the regular expression does not match.

Example

Request from sanitized Tor web server log for virtual host dist.torproject.org from 2018-03-14:

```
0.0.0.1 - - [14/Mar/2018:00:00:00 +0000] "GET /tormessenger/0.5.0b1/tormessenger-insta
```

Step 7: Count Tor Messenger update pings

We count a request as Tor Messenger update ping if it matches the following criteria:

- Request method: GET
- Resource string: `'%/tormessenger/update\__/%'` but none of `'%.xml'`, `'%/'` or `'%/?'`
- Response code: 200

We distinguish platforms based on the resource string: `'%/WINNT%'` for Windows, `'%/Darwin%'` for macOS, and `'%/Linux%'` for Linux.

We extract the locale (for example, `'en-US'` for English as used in the United States or `'de'` for German) from the resource string using regular expression `'.*/([a-zA-Z]{2}|[a-zA-Z]{2}-[a-zA-Z]{2})\??$'`, falling back to `'??'` for unrecognized locales if the regular expression does not match.

Example

Request from sanitized Tor web server log for virtual host aus2.torproject.org from 2018-03-14:

```
0.0.0.1 - - [14/Mar/2018:00:00:00 +0000] "GET /tormessenger/update_2/release/WINNT_x86-
```