Discussion

Why the added complexity from proposal 225?

The complexity difference between this proposal and prop225 is in part because prop225 doesn't specify how the shared random value gets to the clients. This proposal spends lots of effort specifying how the two shared random values can always be readily accessible to clients.

Why do you do a commit-and-reveal protocol in 24 rounds?

The reader might be wondering why we span the protocol over the course of a whole day (24 hours), when only 3 rounds would be sufficient to generate a shared random value.

We decided to do it this way, because we piggyback on the Tor voting protocol which also happens every hour.

We could instead only do the shared randomness protocol from 21:00 to 00:00 every day. Or to do it multiple times a day.

However, we decided that since the shared random value needs to be in every consensus anyway, carrying the commitments/reveals as well will not be a big problem. Also, this way we give more chances for a failing dirauth to recover and rejoin the protocol.

Why can't we recover if the 00:00UTC consensus fails?

If the 00:00UTC consensus fails, there will be no shared random value for the whole day. In theory, we could recover by calculating the shared randomness of the day at 01:00UTC instead. However, the engineering issues with adding such recovery logic are too great. For example, it's not easy for an authority who just booted to learn whether a specific consensus failed to be created.

These lines should be voted on. A majority of votes is sufficient to make a protocol un-supported. A supermajority of authorities (2/3) are needed to make a protocol required. The required protocols should not be locally configurable, but rather should be hardwired in the dirauth implementation.

See Subprotocol versioning for details of how a relay and a client should behave when they encounter these lines in the consensus.

Because of the implications for required-*-protocols (see below), certain safety mechanisms are strongly recommended:

1. An authority SHOULD NOT vote for any protocol in one of these "required" lists that it does not itself support.
2. Authority implementations SHOULD NOT support configurable lists of required protocols: instead, those lists should be hard-coded.

Implications of `required-*-protocols`

> A consensus containing erroneous `required-*-protocols` has the potential to take down the network: clients and relays that see requirements, that they do not meet, will shut down, and not retry.
>
> Note that this applies to versions, not just protocols! If the consensus requires `Foobar=8-9`, and the client only has `Foobar=9`, it will shut down.
>
> For further information, see notes in the C Tor source code.

`params` — Network parameters

- `params` [*Keyword=Value Keyword=Value* ..]
- Distinct _Keyword_s, in lexical order
- At most once

Each ***Keyword*** is a parameter keyword. Each ***Value*** is a signed 32-bit integer, in decimal.

When reading a document, unknown parameters MUST be tolerated.

When a dirauth computes a consensus from votes, every parameter, even one unknown to that dirauth, MUST be included in the consensus if it appears in enough votes.

`shared-rand-*-value` — Shared random values (in consensus)

- `shared-rand-previous-value` *NumReveals Value* ..
- `shared-rand-current-value` *NumReveals Value* ..
- At most once each
- In preamble in consensus only
- (In authority section in votes.)

Value is the shared random value (256 bits) encoded in base64, as calculated by the shared random protocol.

NumReveals is the number of commits used to generate *Value*.

`shared-rand-current-value` is the value that was generated during the latest shared randomness protocol run. `shared-rand-previous-value` is the value generated during the second-to-last run.

> For example, if this document was created on the 5th of November, `current` carries the value generated during the protocol run of the 4th of November, and `previous` carries the value generated during the protocol run of the 3rd of November.
>
> See CONS for why we include old values in votes and consensus.

`bandwidth-file-headers` — Bandwidth file metadata

“bandwidth-file-headers” SP KeyValues NL

[At most once for votes; does not occur in consensuses.]

KeyValues ::= “” | KeyValue | KeyValues SP KeyValue KeyValue ::= Keyword ‘=’ Value Value ::= ArgumentCharValue+ ArgumentCharValue ::= any printing ASCII character except NL and SP.

The headers from the bandwidth file used to generate this vote.

If an authority is not configured with a V3BandwidthsFile, this line SHOULD NOT appear in its vote.

If an authority is configured with a V3BandwidthsFile, but parsing fails, this line SHOULD appear in its vote, but without any headers.

First-appeared: Tor 0.3.5.1-alpha.

reveal). The attacker can also sabotage the consensus, but there are other ways this can be done with the current voting system.

Furthermore, we claim that such an attack is very noisy and detectable. First of all, it requires the authority to sabotage two consensuses which will cause quite some noise. Furthermore, the authority needs to send different votes to different auths which is detectable. Like the commit phase attack, the detection here is to make sure that the commitment values in a vote coming from an authority are always the same for each authority.

Partition attacks

This design is not immune to certain partition attacks. We believe they don't offer much gain to an attacker as they are very easy to detect and difficult to pull off since an attacker would need to compromise a directory authority at the very least. Also, because of the byzantine general problem, it's very hard (even impossible in some cases) to protect against all such attacks. Nevertheless, this section describes all possible partition attack and how to detect them.

Partition attacks during commit phase

A malicious directory authority could send only its commit to one single authority which results in that authority having an extra commit value for the shared random calculation that the others don't have. Since the consensus needs majority, this won't affect the final SRV value. However, the attacker, using this attack, could remove a single directory authority from the consensus decision at 24:00 when the SRV is computed.

An attacker could also partition the authorities by sending two different commitment values to different authorities during the commit phase.

All of the above is fairly easy to detect. Commitment values in the vote coming from an authority should NEVER be different between authorities. If so, this means an attack is ongoing or very bad bug (highly unlikely).

Partition attacks during reveal phase

Let's consider Alice, a malicious directory authority. Alice could wait until the last reveal round, and reveal its value to half of the authorities. That would partition the authorities into two sets: the ones who think that the shared random value should contain this new reveal, and the rest who don't know about it. This would result in a tie and two different shared random value.

A similar attack is possible. For example, two rounds before the end of the reveal phase, Alice could advertise her reveal value to only half of the dirauths. This way, in the last reveal phase round, half of the dirauths will include that reveal value in their votes and the others will not. In the end of the reveal phase, half of the dirauths will calculate a different shared randomness value than the others.

We claim that this attack is not particularly fruitful: Alice ends up having two shared random values to choose from which is a fundamental problem of commit-and-reveal protocols as well (since the last person can always abort or

`bandwidth-file-digest` — Bandwidth file

- `bandwidth-file-digest` *algorithm=digest* ..
- One or more *algorithm=digest* arguments
- At most once
- In votes only

Each ***digest*** is the hash using *algorithm* of the bandwidth file used to generate this vote, in base64, unpadded.

algorithm is `sha256` or another algorithm.

If an authority is not configured with a bandwidth file, this line SHOULD NOT appear in its vote.

If an authority is configured with a bandwidth file, but parsing fails, this line SHOULD appear in its vote, with the digest(s) of the unparseable file.

First-appeared: Tor 0.4.0.4-alpha

> In C Tor this file is configured with `V3BandwidthsFile` in torrc

Vote and consensus document, authority section

The authority section differs between votes and the consensus.

In a consensus, the authority section consists of:

- One or more authority entries, for the actual authorities, in order by authority identity digest.
- Zero or more **superseded authority key entries**, each consisting of only a `dir-source` item.
- No authority key certificates.

In a vote, the authority section gives details of this directory authority, and consists of:

- One authority entry.
- One authority key certificate.

Vote and consensus document, authority entry items

Each authority entry contains the following items. The entries are sorted by authority identity digest.

`dir-source` — Introduces authority entry

- `dir-source` *nickname fingerprint address IP dirport orport* ..
- At start of authority entry, exactly once

Describes this authority.

nickname is a convenient identifier for the authority. ***fingerprint*** is H(KP_auth_id_rsa) as in the `fingerprint` item in the authority's authority key certificate. ***address*** is the server's hostname. ***IP*** is the server's current IP address, and ***dirport*** is its current directory port. ***orport*** is the port at that address where the authority listens for OR connections.

A `dir-source` line for a superseded authority key entry is identical to the entry for that authority, except that:

- *nickname* has `-legacy` appended
- *identity* is the superseded key

`contact` — Authority contact information

- `contact` *string*....
- Exactly once
- *string* is the whole rest of the line

An arbitrary string describing how to contact the directory server's administrator. Administrators should include at least an email address and a PGP fingerprint.

`vote-digest` — Contributing vote

- `vote-digest` *digest* ..
- Exactly once
- In consensus only

The digest of the vote from the authority that contributed to this consensus, as signed (that is, not including the signature). (Hex, upper-case.)

Security Analysis

Security of commit-and-reveal and future directions

The security of commit-and-reveal protocols is well understood, and has certain flaws. Basically, the protocol is insecure to the extent that an adversary who controls b of the authorities gets to choose among 2^b outcomes for the result of the protocol. However, an attacker who is not a dirauth should not be able to influence the outcome at all.

We believe that this system offers sufficient security especially compared to the current situation. More secure solutions require much more advanced crypto and more complex protocols so this seems like an acceptable solution for now.

Here are some examples of possible future directions:

- Schemes based on threshold signatures (e.g. see [HOPPER])
- Unicorn scheme by Lenstra et al. [UNICORN]
- Schemes based on Verifiable Delay Functions [VDFS]

For more alternative approaches on collaborative random number generation also see the discussion at [RNGMESSAGING].

Predicting the shared random value during reveal phase

The reveal phase lasts 12 hours, and most authorities will send their reveal value on the first round of the reveal phase. This means that an attacker can predict the final shared random value about 12 hours before it's generated.

This does not pose a problem for the HSDir hash ring, since we impose an higher uptime restriction on HSDir nodes, so 12 hours predictability is not an issue.

Any other protocols using the shared random value from this system should be aware of this property.

[Exactly once per authority]

The values are the same as detailed in section [COMMITVOTE].

This line is also used by an authority to store its own value.

Finally is the shared random value section.

“SharedRandPreviousValue” SP num_reveals SP value NL

[At most once]

```
        This is the previous shared random value agreed on at the
previous
        period. The fields are the same as in section [SRVOTE].

     "SharedRandCurrentValue" SP num_reveals SP value NL

        [At most once]

        This is the latest shared random value. The fields are the
same as in
        section [SRVOTE].
```

`legacy-dir-key` — Superseded authority key

- `legacy-dir-key` *fingerprint* ..
- At most once
- In votes only

Lists a fingerprint for a superseded identity key still used by this authority to keep older clients working. This optionis used to keep key around for a little while in case the authorities need to migrate many identity keys at once. (Generally, this would only happen because of a security vulnerability that affected multiple authorities, like the Debian OpenSSL RNG bug of May 2008.)

For each `legacy-dir-key` in the vote, there is a supserseded authority key entry in the consensus.

This item does not appear in votes or consensuses, as of May 2025.

`shared-rand-participate` — Indicates shared random participation

- `shared-rand-participate` ..
- At most once
- In votes only

Denotes that the directory authority supports and can participate in the shared random protocol.

`shared-rand-commit` — Shared random commitment

- `shared-rand-commit` *Version AlgName Identity Commit* [Reveal]
- Any number of times
- In votes only

```
Version ::= An integer greater or equal to 0.
AlgName ::= 1\*(ALPHA / DIGIT / "\_" / "-")
Identity ::= 40\* HEXDIG
```

Denotes a directory authority commit for the shared randomness protocol, containing the commitment value and potentially also the reveal value. See sections [COMMITREVEAL] and [VALIDATEVALUES] of srv-spec.txt on how to generate and validate these values.

Version is the current shared randomness protocol version. ***AlgName*** is the hash algorithm that is used (e.g. “sha3-256”) and ***Identity*** is the authority’s SHA1 v3

identity fingerprint. ***Commit*** is the encoded commitment value in base64. ***Reveal*** (optional) contains the reveal value in base64.

If a vote contains multiple commits from the same authority, the receiver MUST only consider the first commit listed.

`shared-rand-*-value` — Shared random values (in votes)

- `shared-rand-previous-value` *NumReveals Value* ..
- `shared-rand-current-value` *NumReveals Value* ..
- In authority section in votes.
- In preamble section in consensus.
- See description in the preamble section

Vote and consensus document, router status entries

Each router status entry contains the following items. Router status entries are sorted in ascending order by identity digest.

`r` — Introduce a router status entry

- `r` *nickname identity digest publication IP ORPort DirPort* ..
- At start of router status entry, exactly once

Nickname is the OR's nickname. ***Identity*** is its SHA1(DER(KP_relayid_rsa)) in unpadded base64. ***Digest*** is a hash of its most recent descriptor as signed (that is, not including the signature) by the RSA identity key (see section 1.3.), encoded in base64.

Publication was once the publication time of the router's most recent descriptor, in the form YYYY-MM-DD HH:MM:SS, in UTC. Now it is only used in votes, and may be set to a fixed value in consensus documents. Implementations SHOULD ignore this value in non-vote documents.

IP is its current IPv4 address. ***ORPort*** is its current OR port. ***DirPort*** is its directory port or `0` for "none".

`a` — Further router address(es) (IPv6)

- `a` *address*:*port*
- Any number

where VALUE is the actual shared random value encoded in hex (computed as specified in section [SRCALC]. NUM_REVEALS is the number of reveal values used to generate this SRV.

To maintain consistent ordering, the shared random values of the previous period should be listed before the values of the current period.

Encoding Shared Random Values in the consensus

Authorities insert the two active shared random values in the consensus following the same encoding format as in [SRVOTE].

Persistent state format

As a way to keep ground truth state in this protocol, an authority MUST keep a persistent state of the protocol. The next sub-section suggest a format for this state which is the same as the current state file format.

It contains a preamble, a commitment and reveal section and a list of shared random values.

The preamble (or header) contains the following items. They MUST occur in the order given here:

"Version" SP version NL

[At start, exactly once.]

A document format version. For this specification, version is "1".

"ValidUntil" SP YYYY-MM-DD SP HH:MM:SS NL

[Exactly once]

```
        After this time, this state is expired and shouldn't be used
nor
        trusted. The validity time period is till the end of the
current
        protocol run (the upcoming noon).
```

The following details the commitment and reveal section. They are encoded the same as in the vote. This makes it easier for implementation purposes.

"Commit" SP version SP algname SP identity SP commit [SP reveal] NL

Validating commitments and reveals

Given a COMMIT message and a REVEAL message it should be possible to verify that they indeed correspond. To do so, the client extracts the random value `SHA3_256(RN)` from the REVEAL message, hashes it, and compares it with the `SHA3_256(SHA3_256(RN))` from the COMMIT message. We say that the COMMIT and REVEAL messages correspond, if the comparison was successful.

Participants MUST also check that corresponding COMMIT and REVEAL values have the same timestamp value.

Authorities should ignore reveal values during the Reveal Phase that don't correspond to commit values published during the Commitment Phase.

Encoding commit/reveal values in votes

An authority puts in its vote the commitments and reveals it has produced and seen from the other authorities. To do so, it includes the following in its votes:

"shared-rand-commit" SP VERSION SP ALGNAME SP IDENTITY SP COMMIT [SP REVEAL] NL

where VERSION is the version of the protocol the commit was created with. IDENTITY is the authority's SHA-1 identity fingerprint (`HEX(SHA1(DER(KP_auth_id_rsa)))`) and COMMIT is the encoded commit [COMMITREVEAL]. Authorities during the reveal phase can also optionally include an encoded reveal value REVEAL. There MUST be only one line per authority else the vote is considered invalid. Finally, the ALGNAME is the hash algorithm that should be used to compute COMMIT and REVEAL which is "sha3-256" for version 1.

Shared Random Value

Authorities include a shared random value (SRV) in their votes using the following encoding for the previous and current value respectively:

```
"shared-rand-previous-value" SP NUM_REVEALS SP VALUE NL
"shared-rand-current-value" SP NUM_REVEALS SP VALUE NL
```

address and ***port*** are as for `or-address`.

Additional reachable address(es) for the OR. Used to convey the OR's IPv6 address.

Clients should ignore any IPv4 addresses in `a`s, and use only the first IPv6 address, if there is one.

Authorities should include only the first advertised IPv6 address, if it is reachable.

> We may extend the protocol in the future to support multiple addresses for each address family.

(Only included when the vote or consensus is generated with consensus-method 14 or later.)

`s` — Router status flags

- `s` [*flag flag* ..]
- Exactly once.
- Zero or more distinct `_flag_` arguments, in lexical order

Each ***flag*** argument states a property of the router. If a particular flag is present in `known-flags`, but absent from the router entry, the document states that the router does *not* have the property in question. If a flag is absent from `known-flags`, information about the property is not available in this document. (Every flag in an `s` item MUST be in `known-flags`.)

Currently specified flags are:

- `Authority` — is a directory authority.
- `BadExit` — is believed to be useless as an exit node (because its ISP censors it, because it is behind a restrictive proxy, or for some similar reason).
- `Exit` — supports commonly used exit ports, and should be treated specially in the path building algorithm.
- `Fast` — is suitable for high-bandwidth circuits.
- `Guard` — is suitable for use as an entry guard.
- `HSDir` — is considered a v2 hidden service directory.
- `MiddleOnly` — is considered unsuitable for usage other than as a middle relay. Since 0.4.7.2-alpha, when it is present, the authorities will automatically vote against flags that would make the router usable in Guard, HSDir, Exit, and V2Dir. Additionally, since Tor 0.4.8.15, clients and services will also avoid usage of MiddleOnly nodes in IP and RP positions.

- `NoEdConsensus` — any Ed25519 key in the router's descriptor or microdescriptor does not reflect authority consensus.
- `Stable` — is suitable for long-lived circuits.
- `StaleDesc` — should upload a new descriptor because the old one is too old.
- `Running` — is currently usable over all its published ORPorts. (Authorities ignore IPv6 ORPorts unless configured to check IPv6 reachability.) Relays without this flag are omitted from the consensus, and current clients (since 0.2.9.4-alpha) assume that every listed relay has this flag.
- `Valid` — has been 'validated'. Clients before 0.2.9.4-alpha would not use routers without this flag by default. Currently, relays without this flag are omitted from the consensus, and current (post-0.2.9.4-alpha) clients assume that every listed relay has this flag.
- `V2Dir` — implements the v2 directory protocol or higher.

Unknown flags must be ignored (in `s` and in `known-flags`) when reading a document. Except, they must be processed by an authority, when generating a consensus from votes.

`v` — Relay's Tor (protocol) version

- `v` *version*....
- At most once
- *version* is the whole rest of the line

The version of the Tor protocol that this relay is running. If *version* begins with `Tor` SP, the rest of the string is a Tor version number, and the protocol is "The Tor protocol as supported by the given version of Tor." Otherwise, Tor has upgraded to a more sophisticated protocol versioning system, and the protocol is "a version of the Tor protocol more recent than any we recognize."

Directory authorities SHOULD omit version strings they receive from descriptors f they would cause `v` lines to be over 128 characters long.

`pr` — Subprotocol capabilities supported

- `pr` *entry entry* ..
- Exactly once
- Syntax and semantics as for `proto` in a server descriptor

When a descriptor does not contain a "proto" entry, the authorities should reconstruct it using the approach described below in section D. They are included in the consensus using the same rules as currently used for "v" lines, if a sufficiently late consensus method is in use.

Specification

Voting

This section describes how commitments, reveals and SR values are encoded in votes. We describe how to encode both the authority's own commitments/reveals and also the commitments/reveals received from the other authorities. Commitments and reveals share the same line, but reveals are optional.

Participating authorities need to include the line:

"shared-rand-participate"

in their votes to announce that they take part in the protocol.

Computing commitments and reveals

A directory authority that wants to participate in this protocol needs to create a new pair of commitment/reveal values for every protocol run. Authorities SHOULD generate a fresh pair of such values right before the first commitment phase of the day (at 00:00UTC).

The value REVEAL is computed as follows:

```
REVEAL = base64-encode( TIMESTAMP || SHA3_256(RN) )

      where RN is the SHA3-256 hashed value of a 256-bit random value.
We hash the
      random value to avoid exposing raw bytes from our PRNG to the
network (see
      [RANDOM-REFS]).

      TIMESTAMP is an 8-bytes network-endian time_t value. Authorities
SHOULD
      set TIMESTAMP to the valid-after time of the vote document they
first plan
      to publish their commit into (so usually at 00:00UTC, except if
they start
      up in a later commit round).

   The value COMMIT is computed as follows:

      COMMIT = base64-encode( TIMESTAMP || SHA3_256(REVEAL) )
```

is, until two shared random values are included in a consensus. This should happen after three 00:00UTC consensuses have been produced, which takes 48 hours.

Rebooting Directory Authorities

The shared randomness protocol must be able to support directory authorities who leave or join in the middle of the protocol execution.

An authority that commits in the Commitment Phase and then leaves MUST have stored its reveal value on disk so that it continues participating in the protocol if it returns before or during the Reveal Phase. The reveal value MUST be stored timestamped to avoid sending it on wrong protocol runs.

An authority that misses the Commitment Phase cannot commit anymore, so it's unable to participate in the protocol for that run. Same goes for an authority that misses the Reveal phase. Authorities who do not participate in the protocol SHOULD still carry commits and reveals of others in their vote.

Finally, authorities MUST implement their persistent state in such a way that they will never commit two different values in the same protocol run, even if they have to reboot in the middle (assuming that their persistent state file is kept). A suggested way to structure the persistent state is found at [STATEFORMAT].

`w` — Bandwidth estimate

- `w` *Keyword=value* ..
- At most once

The following ***Keyword***s are defined:

- `Bandwidth`: Estimated bandwidth of this relay. Used to weight router selection.
- `Measured`: Total measured bandwidth, produced by measuring stream capacities. In votes from bandwidth measurement authorities, only.
- `Unmeasured=1`: Present unless `Bandwidth` is based 3 or more `Measured` values for this relay. In consensus, only.

`Bandwidth` and `Measured` ***`value`***s are 32-bit unsigned integers, in decimal, representing kilobytes per second.

Other weighting keywords may be added later. Unknown keywords (or keywords unexpectedly present in this kind of document) must be ignored.

> When generating a consensus from votes, the consensus method determines precisely which set of keywords are to be recognised/included.

`p` — Exit ports summary

- `p` `accept`|`reject` *PortList*
- At most once

An exit-policy summary summarizing the router's supported exit ports, to "most addresses".

`m` — Microdescriptor hashes

- `m` *methods algorithm=digest* ..
- Any number
- In votes only
- One or more *algorithm=digest* arguments

The `m` items taken together give the microdescriptor hashes, for this router, for each consensus method listed in `consensus-methods`.

methods is a comma-separated list of all the the consensus methods that generate precisely this microdescriptor.

Each ***algorithm*** is the name of the hash algorithm producing *digest*, which can be `sha256` or something else, depending on the consensus "methods" supporting this algorithm. ***digest*** is hash of the microdescriptor in base64, unpadded.

> In consensuses, the microdescriptors appear only in the microdescriptor consensus.

`id` — Relay's (ed25519) identity

- `id ed25519` *KP_relayid_ed*
- `id ed25519 none`
- In votes only
- At most once

KP_relayid_ed is encoded in base64, unpadded.

`stats` — Statistics for this relay

- `stats` [*Keyword=Number Keyword=Number* ..]
- At most once
- In votes only

`Number` has the syntax `[0-9]+(\.[0-9]+)?`.

Various statistics that the authority has computed for this relay.

Reported keys are:

- `wfu` – Weighted Fractional Uptime
- `"tk` – Weighted Time Known
- `mtbf` – Mean Time Between Failure (stability)

(As of tor-0.4.6.1-alpha)

Vote and consensus document, footer

The footer section contains the following items.

> The footer section was omitted prior to consensus method 9.

Shared Random Value Calculation At 00:00UTC

Finally, at 00:00UTC every day, authorities compute a fresh shared random value and this value must be added to the consensus so clients can use it.

Authorities calculate the shared random value using the reveal values in their state as specified in subsection [SRCALC].

Authorities at 00:00UTC start including this new shared random value in their votes, replacing the one from two protocol runs ago. Authorities also start including this new shared random value in the consensus as well.

Apart from that, authorities at 00:00UTC proceed voting normally as they would in the first round of the commitment phase (section [COMMITMENTPHASE]).

Shared Randomness Calculation

An authority that wants to derive the shared random value SRV, should use the appropriate reveal values for that time period and calculate SRV as follows.

```
       HASHED_REVEALS = SHA3_256(ID_a | R_a | ID_b | R_b | ..)

       SRV = SHA3_256("shared-random" | INT_8(REVEAL_NUM) |
INT_4(VERSION) |
                      HASHED_REVEALS | PREVIOUS_SRV)
```

where the ID_a value is the identity key fingerprint of authority 'a' and R_a is the corresponding reveal value of that authority for the current period.

Also, REVEAL_NUM is the number of revealed values in this construction, VERSION is the protocol version number and PREVIOUS_SRV is the previous shared random value. If no previous shared random value is known, then PREVIOUS_SRV is set to 32 NUL (\x00) bytes.

To maintain consistent ordering in HASHED_REVEALS, all the ID_a | R_a pairs are ordered based on the R_a value in ascending order.

Bootstrapping Procedure

As described in [CONS], two shared random values are required for the HSDir overlay periods to work properly as specified in proposal 224. Hence clients MUST NOT use the randomness of this system till it has bootstrapped completely; that

Persistent State During Commitment Phase

During the commitment phase, authorities save in their persistent state the authoritative commits they have received from each authority. Only one commit per authority must be considered trusted and active at a given time.

Reveal Phase

The reveal phase lasts from 12:00UTC to 00:00UTC.

Now that the commitments have been agreed on, it's time for authorities to reveal their random values.

Voting During Reveal Phase

During the reveal phase, each authority includes in its votes:

```
    - Its reveal value that was previously committed in the commit
phase.
    - All the commitments and reveals received from other authorities.
    - The two previous shared random values produced by the protocol
(if any).
```

The set of commitments have been decided during the commitment phase and must remain the same. If an authority tries to change its commitment during the reveal phase or introduce a new commitment, the new commitment MUST be ignored.

Persistent State During Reveal Phase

During the reveal phase, authorities keep the authoritative commits from the commit phase in their persistent state. They also save any received reveals that correspond to authoritative commits and are valid (as specified in [VALIDATEVALUES]).

An authority that just received a reveal value from another authority's vote, MUST wait till the next voting round before including that reveal value in its votes.

`directory-footer` — Introduce directory footer

- `directory-footer`
- No extra arguments
- Exactly once, at start of the footer

`bandwidth-weights` — Weights to apply during path selection

- `bandwidth-weights` [*Keyword=Int32 Keyword=Int32* ..]
- At most once
- In consensus only
- `Int32` is a decimal integer between -2147483648 and 2147483647.

List of optional weights to apply to router bandwidths during path selection. They are sorted in lexical order and values are divided by the consensus' "bwweightscale" param. Definition of our known entries are...

- `Wgg` – Weight for Guard-flagged nodes in the guard position
- `Wgm` – Weight for non-flagged nodes in the guard Position
- `Wgd` – Weight for Guard+Exit-flagged nodes in the guard Position
- `Wmg` – Weight for Guard-flagged nodes in the middle Position
- `Wmm` – Weight for non-flagged nodes in the middle Position
- `Wme` – Weight for Exit-flagged nodes in the middle Position
- `Wmd` – Weight for Guard+Exit flagged nodes in the middle Position
- `Weg` – Weight for Guard flagged nodes in the exit Position
- `Wem` – Weight for non-flagged nodes in the exit Position
- `Wee` – Weight for Exit-flagged nodes in the exit Position
- `Wed` – Weight for Guard+Exit-flagged nodes in the exit Position
- `Wgb` – Weight for BEGIN_DIR-supporting Guard-flagged nodes
- `Wmb` – Weight for BEGIN_DIR-supporting non-flagged nodes
- `Web` – Weight for BEGIN_DIR-supporting Exit-flagged nodes
- `Wdb` – Weight for BEGIN_DIR-supporting Guard+Exit-flagged nodes
- `Wbg` – Weight for Guard flagged nodes for BEGIN_DIR requests
- `Wbm` – Weight for non-flagged nodes for BEGIN_DIR requests
- `Wbe` – Weight for Exit-flagged nodes for BEGIN_DIR requests
- `Wbd` – Weight for Guard+Exit-flagged nodes for BEGIN_DIR requests

These values are calculated as specified in section 3.8.3.

Vote and consensus document, signatures

The Signature Item is:

`directory-signature` **— Signature**

- `directory-signature` [*Algorithm*] *identity signing-key-digest* ..
- *Signature*, Object, `SIGNATURE`
- Exactly once in votes
- At least once in consensus

This is a signature of the status document, with the initial item “network-status-version”, and the signature item “directory-signature”, using the signing key. (In this case, we take the hash through the *space* after directory-signature, not the newline: this ensures that all authorities sign the same thing.) “identity” is the hex-encoded digest of the authority identity key of the signing authority, and “signing-key-digest” is the hex-encoded digest of the current authority signing key of the signing authority.

The Algorithm is one of “sha1” or “sha256” if it is present; implementations MUST ignore directory-signature entries with an unrecognized Algorithm. “sha1” is the default, if no Algorithm is given. The algorithm describes how to compute the hash of the document before signing it.

“ns”-flavored consensus documents must contain only sha1 signatures. Votes and microdescriptor documents may contain other signature types. Note that only one signature from each authority should be “counted” as meaning that the authority has signed the consensus.

(Tor clients before 0.2.3.x did not understand the ‘algorithm’ field.)

Protocol

In this section we give a detailed specification of the protocol. We describe the protocol participants’ logic and the messages they send. The encoding of the messages is specified in the next section ([SPEC]).

Now we go through the phases of the protocol:

Commitment Phase

The commit phase lasts from 00:00UTC to 12:00UTC.

During this phase, an authority commits a value in its vote and saves it to the permanent state as well.

Authorities also save any received authoritative commits by other authorities in their permanent state. We call a commit by Alice “authoritative” if it was included in Alice’s vote.

Voting During Commitment Phase

During the commit phase, each authority includes in its votes:

```
     - The commitment value for this protocol run.
     - Any authoritative commitments received from other authorities.
     - The two previous shared random values produced by the protocol
 (if any).
```

The commit phase lasts for 12 hours, so authorities have multiple chances to commit their values. An authority MUST NOT commit a second value during a subsequent round of the commit phase.

If an authority publishes a second commitment value in the same commit phase, only the first commitment should be taken in account by other authorities. Any subsequent commitments MUST be ignored.

During the commitment phase, it is populated with the commitments of all authorities. Then during the reveal phase, the reveal values are also stored in the state.

As discussed previously, the shared random values from the current and previous time period must also be present in the state at all times if they are available.

Protocol Illustration

An illustration for better understanding the protocol can be found here:

https://people.torproject.org/~asn/hs_notes/shared_rand.jpg

It reads left-to-right.

The illustration displays what the authorities (A_1, A_2, A_3) put in their votes. A chain 'A_1 -> c_1 -> r_1' denotes that authority A_1 committed to the value c_1 which corresponds to the reveal value r_1.

The illustration depicts only a few rounds of the whole protocol. It starts with the first three rounds of the commit phase, then it jumps to the last round of the commit phase. It continues with the first two rounds of the reveal phase and then it jumps to the final round of the protocol run. It finally shows the first round of the commit phase of the next protocol run (00:00UTC) where the final Shared Random Value is computed. In our fictional example, the SRV was computed with 3 authority contributions and its value is "a56fg39h".

We advice you to revisit this after you have read the whole document.

Assigning flags in a vote

(This section describes how directory authorities choose which status flags to apply to routers. Later directory authorities MAY do things differently, so long as clients keep working well. Clients MUST NOT depend on the exact behaviors in this section.)

In the below definitions, a router is considered "active" if it is running, valid, and not hibernating.

When we speak of a router's bandwidth in this section, we mean either its measured bandwidth, or its advertised bandwidth. If a sufficient threshold (configurable with MinMeasuredBWsForAuthToIgnoreAdvertised, 500 by default) of routers have measured bandwidth values, then the authority bases flags on *measured* bandwidths, and treats nodes with non-measured bandwidths as if their bandwidths were zero. Otherwise, it uses measured bandwidths for nodes that have them, and advertised bandwidths for other nodes.

When computing thresholds based on percentiles of nodes, an authority only considers nodes that are active, that have not been omitted as a sybil (see below), and whose bandwidth is at least 4 KB. Nodes that don't meet these criteria do not influence any threshold calculations (including calculation of stability and uptime and bandwidth thresholds) and also do not have their Exit status change.

"Valid" – a router is 'Valid' if it is running a version of Tor not known to be broken, and the directory authority has not blacklisted it as suspicious.

```
"Named" --
"Unnamed" -- Directory authorities no longer assign these flags.
   They were once used to determine whether a relay's nickname was
   canonically linked to its public key.
```

"Running" – A router is 'Running' if the authority managed to connect to it successfully within the last 45 minutes on all its published ORPorts. Authorities check reachability on:

```
  * the IPv4 ORPort in the "r" line, and
  * the IPv6 ORPort considered for the "a" line, if:
    * the router advertises at least one IPv6 ORPort, and
    * AuthDirHasIPv6Connectivity 1 is set on the authority.
```

A minority of voting authorities that set AuthDirHasIPv6Connectivity will drop unreachable IPv6 ORPorts from the full consensus. Consensus method 27 in 0.3.3.x puts IPv6 ORPorts in the microdesc consensus, so that authorities can drop unreachable IPv6 ORPorts from all consensus flavors. Consensus method 28 removes IPv6 ORPorts from microdescriptors.

“Stable” – A router is ‘Stable’ if it is active, and either its Weighted MTBF is at least the median for known active routers or its Weighted MTBF corresponds to at least 7 days. Routers are never called Stable if they are running a version of Tor known to drop circuits stupidly. (0.1.1.10-alpha through 0.1.1.16-rc are stupid this way.)

To calculate weighted MTBF, compute the weighted mean of the lengths of all intervals when the router was observed to be up, weighting intervals by $ \alpha^n$, where n is the amount of time that has passed since the interval ended, and α is chosen so that measurements over approximately one month old no longer influence the weighted MTBF much.

[XXXX what happens when we have less than 4 days of MTBF info.]

“Exit” – A router is called an ‘Exit’ iff it allows exits to at least one /8 address space on each of ports 80 and 443. (Up until Tor version 0.3.2, the flag was assigned if relays exit to at least two of the ports 80, 443, and 6667.)

“Fast” – A router is ‘Fast’ if it is active, and its bandwidth is either in the top 7/8ths for known active routers or at least 100KB/s.

“Guard” – A router is a possible Guard if all of the following apply:

```
       - It is Fast,
       - It is Stable,
       - Its Weighted Fractional Uptime is at least the median for
"familiar"
         active routers,
       - It is "familiar",
       - Its bandwidth is at least AuthDirGuardBWGuarantee (if set, 2
MB by
         default), OR its bandwidth is among the 25% fastest relays,
       - It qualifies for the V2Dir flag as described below (this
         constraint was added in 0.3.3.x, because in 0.3.0.x clients
         started avoiding guards that didn't also have the V2Dir
flag).
```

To calculate weighted fractional uptime, compute the fraction of time that the router is up in any given day, weighting so that downtime and uptime in the past counts less.

Inserting Shared Random Values in the consensus

After voting happens, we need to be careful on how we pick which shared random values (SRV) to put in the consensus, to avoid breaking the consensus because of authorities having different views of the commit-and-reveal protocol (because maybe they missed some rounds of the protocol).

For this reason, authorities look at the received votes before creating a consensus and employ the following logic:

```
   - First of all, they make sure that the agreed upon consensus
method is
     above the SR consensus method.

   - Authorities include an SRV in the consensus if and only if the
SRV has
     been voted by at least the majority of authorities.

   - For the consensus at 00:00UTC, authorities include an SRV in the
consensus
     if and only if the SRV has been voted by at least
AuthDirNumAgreements
     authorities (where AuthDirNumAgreements is a newly introduced
consensus
     parameter).
```

Authorities include in the consensus the most popular SRV that also satisfies the above constraints. Otherwise, no SRV should be included.

The above logic is used to make it harder to break the consensus by natural partioning causes.

We use the AuthDirNumAgreements consensus parameter to enforce that a *supermajority* of dirauths supports the SR protocol during SRV creation, so that even if a few of those dirauths drop offline in the middle of the run the SR protocol does not get disturbed. We go to extra lengths to ensure this because changing SRVs in the middle of the day has terrible reachability consequences for hidden service clients.

Persistent State of the Protocol

A directory authority needs to keep a persistent state on disk of the on going protocol run. This allows an authority to join the protocol seamlessly in the case of a reboot.

```
        Starting at 00:00UTC and for a period of 12 hours, authorities
every
        hour include their commitment in their votes. They also
include any
        received commitments from other authorities, if available.

      Reveal phase:

        At 12:00UTC, the reveal phase starts and lasts till the end of
the
        protocol at 00:00UTC. In this stage, authorities must reveal
the value
        they committed to in the previous phase. The commitment and
revealed
        values from other authorities, when available, are also added
to the
        vote.

      Shared Randomness Calculation:

        At 00:00UTC, the shared random value is computed from the
agreed
        revealed values and added to the consensus.

   This concludes the commit-and-reveal protocol every day at
00:00UTC.
```

How we use the consensus

The produced shared random values need to be readily available to clients. For this reason we include them in the consensus documents.

Every hour the consensus documents need to include the shared random value of the day, as well as the shared random value of the previous day. That's because either of these values might be needed at a given time for a Tor client to access a hidden service according to section [TIME-OVERLAP] of proposal 224. This means that both of these two values need to be included in votes as well.

Hence, consensuses need to include:

```
      (a) The shared random value of the current time period.
      (b) The shared random value of the previous time period.
```

For this, a new SR consensus method will be needed to indicate which authorities support this new protocol.

A node is 'familiar' if 1/8 of all active nodes have appeared more recently than it, OR it has been around for a few weeks.

"Authority" – A router is called an 'Authority' if the authority generating the network-status document believes it is an authority.

"V2Dir" – A router supports the v2 directory protocol or higher if it has an open directory port OR a tunnelled-dir-server line in its router descriptor, and it is running a version of the directory protocol that supports the functionality clients need. (Currently, every supported version of Tor supports the functionality that clients need, but some relays might set "DirCache 0" or set really low rate limiting, making them unqualified to be a directory mirror, i.e. they will omit the tunnelled-dir-server line from their descriptor.)

"HSDir" – A router is a v2 hidden service directory if it stores and serves v2 hidden service descriptors, has the Stable and Fast flag, and the authority believes that it's been up for at least 96 hours (or the current value of MinUptimeHidServDirectoryV2).

"MiddleOnly" – An authority should vote for this flag if it believes that a relay is unsuitable for use except as a middle relay. When voting for this flag, the authority should also vote against "Exit", "Guard", "HsDir", and "V2Dir". When voting for this flag, if the authority votes on the "BadExit" flag, the authority should vote in favor of "BadExit". (This flag was added in 0.4.7.2-alpha.)

"NoEdConsensus" – authorities should not vote on this flag; it is produced as part of the consensus for consensus method 22 or later.

"StaleDesc" – authorities should vote to assign this flag if the published time on the descriptor is over 18 hours in the past. (This flag was added in 0.4.0.1-alpha.)

"Sybil" – authorities SHOULD NOT accept more than 2 relays on a single IP. If this happens, the authority *should* vote for the excess relays, but should omit the Running or Valid flags and instead should assign the "Sybil" flag. When there are more than 2 (or AuthDirMaxServersPerAddr) relays to choose from, authorities should first prefer authorities to non-authorities, then prefer Running to non-Running, and then prefer high-bandwidth to low-bandwidth relays. In this comparison, measured bandwidth is used unless it is not present for a router, in which case advertised bandwidth is used.

Thus, the network-status vote includes all non-blacklisted, non-expired, non-superseded descriptors.

The bandwidth in a “w” line should be taken as the best estimate of the router’s actual capacity that the authority has. For now, this should be the lesser of the observed bandwidth and bandwidth rate limit from the server descriptor. It is given in kilobytes per second, and capped at some arbitrary value (currently 10 MB/s).

The Measured= keyword on a “w” line vote is currently computed by multiplying the previous published consensus bandwidth by the ratio of the measured average node stream capacity to the network average. If 3 or more authorities provide a Measured= keyword for a router, the authorities produce a consensus containing a “w” Bandwidth= keyword equal to the median of the Measured= votes.

As a special case, if the “w” line in a vote is about a relay with the Authority flag, it should not include a Measured= keyword. The goal is to leave such relays marked as Unmeasured, so they can reserve their attention for authority-specific activities. “w” lines for votes about authorities may include the bandwidth authority’s measurement using a different keyword, e.g. MeasuredButAuthority=, so it can still be reported and recorded for posterity.

The ports listed in a “p” line should be taken as those ports for which the router’s exit policy permits ‘most’ addresses, ignoring any accept not for all addresses, ignoring all rejects for private netblocks. “Most” addresses are permitted if no more than 2^25 IPv4 addresses (two /8 networks) were blocked. The list is encoded as described in section 3.8.2.

Overview

This proposal alters the Tor consensus protocol such that a random number is generated every midnight by the directory authorities during the regular voting process. The distributed random generator scheme is based on the commit-and-reveal technique.

The proposal also specifies how the final shared random value is embedded in consensus documents so that clients who need it can get it.

Introduction to our commit-and-reveal protocol

Every day, before voting for the consensus at 00:00UTC each authority generates a new random value and keeps it for the whole day. The authority cryptographically hashes the random value and calls the output its “commitment” value. The original random value is called the “reveal” value.

The idea is that given a reveal value you can cryptographically confirm that it corresponds to a given commitment value (by hashing it). However given a commitment value you should not be able to derive the underlying reveal value. The construction of these values is specified in section [COMMITREVEAL].

Ten thousand feet view of the protocol

Our commit-and-reveal protocol aims to produce a fresh shared random value (denoted shared_random_value here and elsewhere) every day at 00:00UTC. The final fresh random value is embedded in the consensus document at that time.

Our protocol has two phases and uses the hourly voting procedure of Tor. Each phase lasts 12 hours, which means that 12 voting rounds happen in between. In short, the protocol works as follows:

Commit phase:

Introduction

Motivation

For the next generation hidden services project, we need the Tor network to produce a fresh random value every day in such a way that it cannot be predicted in advance or influenced by an attacker.

Currently we need this random value to make the HSDir hash ring unpredictable (#8244), which should resolve a wide class of hidden service DoS attacks and should make it harder for people to gauge the popularity and activity of target hidden services. Furthermore this random value can be used by other systems in need of fresh global randomness like Tor-related protocols (e.g. OnioNS) or even non-Tor-related (e.g. warrant canaries).

Previous work

Proposal 225 specifies a commit-and-reveal protocol that can be run as an external script and have the results be fed to the directory authorities. However, directory authority operators feel unsafe running a third-party script that opens TCP ports and accepts connections from the Internet. Hence, this proposal aims to embed the commit-and-reveal idea in the Tor voting process which should make it smoother to deploy and maintain.

Serving bandwidth list files

> Note that the URLs here have variants ending in ".z".

If an authority has used a bandwidth list file to generate a vote document it SHOULD make it available at

`http://<hostname>/tor/status-vote/next/bandwidth`

at the start of each voting period.

It MUST NOT attempt to send its bandwidth list file in a HTTP POST to other authorities and it SHOULD NOT make bandwidth list files from other authorities available.

If an authority makes this file available, it MUST be the bandwidth file used to create the vote document available at

`http://<hostname>/tor/status-vote/next/authority`

To avoid inconsistent reads, authorities SHOULD only read the bandwidth file once per voting period. Further processing and serving SHOULD use a cached copy.

The bandwidth list format is described in bandwidth-file-spec.txt.

The standard URLs for bandwidth list files first-appeared in Tor 0.4.0.4-alpha.

Downloading information from other directory authorities

Downloading missing certificates from other directory authorities

XXX when to download certificates.

Downloading server descriptors from other directory authorities

Periodically (currently, every 10 seconds), directory authorities check whether there are any specific descriptors that they do not have and that they are not currently trying to download. Authorities identify them by hash in vote (if publication date is more recent than the descriptor we currently have).

[XXXX need a way to fetch descriptors ahead of the vote? v2 status docs can do that for now.]

If so, the directory authority launches requests to the authorities for these descriptors, such that each authority is only asked for descriptors listed in its most recent vote. If more than one authority lists the descriptor, we choose which to ask at random.

If one of these downloads fails, we do not try to download that descriptor from the authority that failed to serve it again unless we receive a newer network-status (consensus or vote) from that authority that lists the same descriptor.

```
   Directory authorities must potentially cache multiple descriptors
for each
   router. Authorities must not discard any descriptor listed by any
recent
   consensus.  If there is enough space to store additional
descriptors,
   authorities SHOULD try to hold those which clients are likely to
download the
   most.  (Currently, this is judged based on the interval for which
each
   descriptor seemed newest.)
[XXXX define recent]
```

Tor Shared Random Subsystem Specification

This document specifies how the commit-and-reveal shared random subsystem of Tor works. This text used to be proposal 250-commit-reveal-consensus.txt.

Limited ed diff format

We support the following format for consensus diffs. It's a subset of the ed diff format, but clients MUST NOT accept other ed commands.

We support the following ed commands, each on a line by itself:

```
    - "<n1>d"          Delete line n1
    - "<n1>,<n2>d"     Delete lines n1 through n2, inclusive
    - "<n1>,$d"        Delete line n1 through the end of the file,
inclusive.
    - "<n1>c"          Replace line n1 with the following block
    - "<n1>,<n2>c"     Replace lines n1 through n2, inclusive, with
the
                       following block.
    - "<n1>a"          Append the following block after line n1.
```

Note that line numbers always apply to the file after all previous commands have already been applied. Note also that line numbers are 1-indexed.

The commands MUST apply to the file from back to front, such that lines are only ever referred to by their position in the original file.

If there are any directory signatures on the original document, the first command MUST be a "`<n1>,$d`" form to remove all of the directory signatures. Using this format ensures that the client will successfully apply the diff even if they have an unusual encoding for the signatures.

The replace and append command take blocks. These blocks are simply appended to the diff after the line with the command. A line with just a period (".") ends the block (and is not part of the lines to add). Note that it is impossible to insert a line with just a single dot.

Authorities SHOULD NOT download descriptors for routers that they would immediately reject for reasons listed in section 3.2.

Downloading extra-info documents from other directory authorities

Periodically, an authority checks whether it is missing any extra-info documents: in other words, if it has any server descriptors with an extra-info-digest field that does not match any of the extra-info documents currently held. If so, it downloads whatever extra-info documents are missing. We follow the same splitting and back-off rules as in section 3.6.

Computing a consensus from a set of votes

> Note that many of the URLs here have variants ending in ".z".

Given a set of votes, authorities compute the contents of the consensus.

The consensus status, along with as many signatures as the server currently knows (see section 3.10 below), should be available at

`http://<hostname>/tor/status-vote/next/consensus`

The contents of the consensus document are as follows:

The "valid-after", "valid-until", and "fresh-until" times are taken as the median of the respective values from all the votes.

The times in the "voting-delay" line are taken as the median of the VoteSeconds and DistSeconds times in the votes.

Known-flags is the union of all flags known by any voter.

Entries are given on the "params" line for every keyword on which a majority of authorities (total authorities, not just those participating in this vote) voted on, or if at least three authorities voted for that parameter. The values given are the low-median of all votes on that keyword.

(In consensus methods 7 to 11 inclusive, entries were given on the "params" line for every keyword on which *any* authority voted, the value given being the low-median of all votes on that keyword.)

```
     "client-versions" and "server-versions" are sorted in ascending
      order; A version is recommended in the consensus if it is
 recommended
      by more than half of the voting authorities that included a
      client-versions or server-versions lines in their votes.
```

With consensus methods 19 through 33, a package line is generated for a given PACKAGENAME/VERSION pair if at least three authorities list such a package in their votes. (Call these lines the "input" lines for PACKAGENAME.) The consensus will contain every "package" line that is listed verbatim by more than half of the authorities listing a line for the PACKAGENAME/VERSION pair, and no others.

Inferring missing proto lines

The directory authorities no longer allow versions of Tor before 0.2.4.18-rc. But right now, there is no version of Tor in the consensus before 0.2.4.19. Therefore, we should disallow versions of Tor earlier than 0.2.4.19, so that we can have the protocol list for all current Tor versions include:

Cons=1-2 Desc=1-2 DirCache=1 HSDir=1 HSIntro=3 HSRend=1-2 Link=1-4 LinkAuth=1 Microdesc=1-2 Relay=1-2

For Desc, Microdesc and Cons, Tor versions before 0.2.7.stable should be taken to only support version 1.

[Recomputing the sign bit from the private key every time sounds rather strange and inefficient to me... —isis]

Note that in addition to its coordinates, an expanded Ed25519 private key also has a 32-byte random value, “prefix”, used to compute internal `r` values in the signature. For security, this prefix value should be derived deterministically from the curve25519 key. The Tor implementation derives it as `SHA512(private_key | STR)[0..32]`, where STR is the nul-terminated string:

“Derive high part of ed25519 key from curve25519 key\0”

On the client side, where there is no access to the curve25519 private keys, one may use the curve25519 public key’s Montgomery u-coordinate to recover the Montgomery v-coordinate by computing the right-hand side of the Montgomery curve equation:

bv^2 = u(u^2 + au +1)

where

```
a = 486662
b = 1
```

Then, knowing the intended sign of the Edwards x-coordinate, one may recover said x-coordinate by computing:

x = (u/v) * sqrt(-a - 2)

The authority item groups (dir-source, contact, fingerprint, vote-digest) are taken from the votes of the voting authorities. These groups are sorted by the digests of the authorities identity keys, in ascending order. If the consensus method is 3 or later, a dir-source line must be included for every vote with legacy-key entry, using the legacy-key’s fingerprint, the voter’s ordinary nickname with the string “-legacy” appended, and all other fields as from the original vote’s dir-source line.

```
     A router status entry:
        * is included in the result if some router status entry with
the same
          identity is included by more than half of the authorities
(total
          authorities, not just those whose votes we have).
          (Consensus method earlier than 21)

        * is included according to the rules in section 3.8.0.1 and
          3.8.0.2 below. (Consensus method 22 or later)

        * For any given RSA identity digest, we include at most
          one router status entry.

        * For any given Ed25519 identity, we include at most one
router
          status entry.

        * A router entry has a flag set if that is included by more
than half
          of the authorities who care about that flag.

        * Two router entries are "the same" if they have the same
          (descriptor digest, published time, nickname, IP, ports>
tuple.
          We choose the tuple for a given router as whichever tuple
appears
          for that router in the most votes.  We break ties first in
favor of
          the more recently published, then in favor of smaller server
          descriptor digest.

       [
        * The Named flag appears if it is included for this
routerstatus by
          _any_ authority, and if all authorities that list it list
the same
          nickname. However, if consensus-method 2 or later is in use,
and
          any authority calls this identity/nickname pair Unnamed,
then
          this routerstatus does not get the Named flag.

        * If consensus-method 2 or later is in use, the Unnamed flag
is
          set for a routerstatus if any authorities have voted for a
different
          identities to be Named with that nickname, or if any
authority
          lists that nickname/ID pair as Unnamed.

          (With consensus-method 1, Unnamed is set like any other
flag.)
```

Converting a curve25519 public key to an ed25519 public key

Given an X25519 key, that is, an affine point (u,v) on the Montgomery curve defined by

bv^2 = u(u^2 + au +1)

where

```
a = 486662
b = 1
```

and comprised of the compressed form (i.e. consisting of only the u-coordinate), we can retrieve the y-coordinate of the affine point (x,y) on the twisted Edwards form of the curve defined by

-x^2 + y^2 = 1 + d x^2 y^2

where

d = - 121665/121666

by computing

y = (u-1)/(u+1).

and then we can apply the usual curve25519 twisted Edwards point decompression algorithm to find *an* x-coordinate of an affine twisted Edwards point to check signatures with. Signing keys for ed25519 are compressed curve points in twisted Edwards form (so a y-coordinate and the sign of the x-coordinate), and X25519 keys are compressed curve points in Montgomery form (i.e. a u-coordinate).

However, note that compressed point in Montgomery form neglects to encode what the sign of the corresponding twisted Edwards x-coordinate would be. Thus, we need the sign of the x-coordinate to do this operation; otherwise, we'll have two possible x-coordinates that might have correspond to the ed25519 public key.

To get the sign, the easiest way is to take the corresponding private key, feed it to the ed25519 public key generation algorithm, and see what the sign is.

Microdescriptors are available at:

`http://<hostname>/tor/micro/d/<D1>-<D2>-<D3>`

whereas `<Dn>` are base64 encoded SHA256 hashes of the microdescriptors with the trailing =s omitted. -s are used instead of +s to separate items, since the + character is used in base64 encoding. (See also Downloading microdescriptors

Extra-info documents are available at the URLS:

```
http://<hostname>/tor/extra/d/...
http://<hostname>/tor/extra/fp/...
http://<hostname>/tor/extra/all
http://<hostname>/tor/extra/authority
```

(These work like the `/tor/server/` URLs: they support fetching extra-info documents by their SHA1 digest, by the fingerprint of their servers, or all at once. When serving by fingerprint, we serve the extra-info that corresponds to the descriptor we would serve by that fingerprint. Only directory authorities of version 0.2.0.1-alpha or later are guaranteed to support the first three classes of URLs. Caches may support them, and MUST support them if they have advertised "caches-extra-info".)

Clients SHOULD use upper case letters (A-F) when base16-encoding fingerprints. Servers MUST accept both upper and lower case fingerprints in requests.

```
          [But note that authorities no longer vote for the Named
flag,
          and the above two bulletpoints are now irrelevant.]
        ]

        * The version is given as whichever version is listed by the
most
          voters, with ties decided in favor of more recent versions.

        * If consensus-method 4 or later is in use, then routers that
          do not have the Running flag are not listed at all.

        * If consensus-method 5 or later is in use, then the "w" line
          is generated using a low-median of the bandwidth values from
          the votes that included "w" lines for this router.

        * If consensus-method 5 or later is in use, then the "p" line
          is taken from the votes that have the same policy summary
          for the descriptor we are listing.  (They should all be the
          same.  If they are not, we pick the most commonly listed
          one, breaking ties in favor of the lexicographically larger
          vote.)  The port list is encoded as specified in section
3.8.2.

        * If consensus-method 6 or later is in use and if 3 or more
          authorities provide a Measured= keyword in their votes for
          a router, the authorities produce a consensus containing a
          Bandwidth= keyword equal to the median of the Measured=
votes.

        * If consensus-method 7 or later is in use, the params line is
          included in the output.

        * If the consensus method is under 11, bad exits are
considered as
          possible exits when computing bandwidth weights.  Otherwise,
if
          method 11 or later is in use, any router that is determined
to get
          the BadExit flag doesn't count when we're calculating
weights.

        * If consensus method 12 or later is used, only consensus
          parameters that more than half of the total number of
          authorities voted for are included in the consensus.

        [ As of 0.2.6.1-alpha, authorities no longer advertise or
negotiate
          any consensus methods lower than 13. ]

        * If consensus method 13 or later is used, microdesc
consensuses
          omit any router for which no microdesc was agreed upon.

        * If consensus method 14 or later is used, the ns consensus
```

```
and
          microdescriptors may include an "a" line for each router,
listing
          an IPv6 OR port.

        * If consensus method 15 or later is used, microdescriptors
          include "p6" lines including IPv6 exit policies.

        * If consensus method 16 or later is used, ntor-onion-key
          are included in microdescriptors

        * If consensus method 17 or later is used, authorities impose
a
          maximum on the Bandwidth= values that they'll put on a 'w'
          line for any router that doesn't have at least 3 measured
          bandwidth values in votes. They also add an "Unmeasured=1"
          flag to such 'w' lines.

        * If consensus method 18 or later is used, authorities include
          "id" lines in microdescriptors. This method adds RSA ids.

        * If consensus method 19 or later is used, authorities may
include
          "package" lines in consensuses.

        * If consensus method 20 or later is used, authorities may
include
          GuardFraction information in microdescriptors.

        * If consensus method 21 or later is used, authorities may
include
          an "id" line for ed25519 identities in microdescriptors.

        [ As of 0.2.8.2-alpha, authorities no longer advertise or
negotiate
          consensus method 21, because it contains bugs. ]

        * If consensus method 22 or later is used, and the votes do
not
          produce a majority consensus about a relay's Ed25519 key
(see
          3.8.0.1 below), the consensus must include a NoEdConsensus
flag on
          the "s" line for every relay whose listed Ed key does not
reflect
          consensus.

        * If consensus method 23 or later is used, authorities include
          shared randomness protocol data on their votes and
consensus.

        * If consensus-method 24 or later is in use, then routers that
          do not have the Valid flag are not listed at all.

        [ As of 0.3.4.1-alpha, authorities no longer advertise or
```

The most recent descriptors for relays with identity fingerprints `<F1>`, `<F2>`,`<F3>` should be available at:

`http://<hostname>/tor/server/fp/<F1>+<F2>+<F3>`

(NOTE: Due to squid proxy url limitations at most 96 fingerprints can be retrieved in a single request.

Implementations SHOULD NOT download descriptors by identity key fingerprint. This allows a corrupted server (in collusion with a cache) to provide a unique descriptor to a client, and thereby partition that client from the rest of the network.)

The server descriptor with SHA1 (descriptor) digest `<D>` (in hex) should be available at:

`http://<hostname>/tor/server/d/<D>`

The most recent descriptors with SHA1 digests `<D1>`, `<D2>`, `<D3>` should be available at:

`http://<hostname>/tor/server/d/<D1>+<D2>+<D3>.z`

The most recent router descriptor for this server should be at:

`http://<hostname>/tor/server/authority`

This is used by authorities, and also if a server is configured as a bridge. The official Tor implementations (starting at 0.1.1.x) use this resource to test whether a server's own DirPort is reachable. It is also useful for debugging purposes.

> The path element `authority` here is misleading. This is nothing to do with directory authorities. It's the server's own router descriptor, `authority` refers here merely to the fact that this server is (obviously) authoritative for its own descriptor.

A concatenated set of the most recent available descriptors for all known servers should be available at:

`http://<hostname>/tor/server/all`

> `all` is used for archiving and monitoring. Descriptors may be missing, for example descriptors for relays recently added to the consensus, which the directory server hasn't yet managed to obtained.

Clients SHOULD use this format when requesting consensus documents from directory authority servers and from caches running a version of Tor that is known to support this URL format.

Consensus documents are also available as diffs by specifying a `X-Or-Diff-From-Consensus` header, or by fetching from:

```
http://<hostname>/tor/status-vote/current/consensus/diff/<HASH>/
<FPRLIST>
http://<hostname>/tor/status-vote/current/consensus-<FLAVOR>/diff/
<HASH>/<FPRLIST>
```

A concatenated set of all the current authority key certificates (from the current consensus) should be available at:

```
http://<hostname>/tor/keys/all
```

The authority key certificate for this authority should be available at:

```
http://<hostname>/tor/keys/authority
```

The authority key certificate for an authority whose authority identity fingerprint (`HEX(SHA1(DER(KP_auth_id_rsa)))`) is `<F>` should be available at:

```
http://<hostname>/tor/keys/fp/<F>
```

The authority key certificate whose signing key fingerprint (`HEX(SHA1(DER(KP_auth_sign_rsa)))`) is `<F>` should be available at:

```
http://<hostname>/tor/keys/sk/<F>
```

The authority key certificate whose authority identity key fingerprint is `<F>` and whose signing key fingerprint is `<S>` should be available at:

```
http://<hostname>/tor/keys/fp-sk/<F>-<S>
```

(As usual, clients may request multiple certificates using:

```
http://<hostname>/tor/keys/fp-sk/<F1>-<S1>+<F2>-<S2> )
```

[The above fp-sk format was not supported before Tor 0.2.1.9-alpha.]

The most recent descriptor for a relay whose identity key has a fingerprint (`HEX(SHA1(DER(KP_relayid_rsa)))`) of `<F>` should be available at:

```
http://<hostname>/tor/server/fp/<F>
```

```
negotiate
          any consensus methods lower than 25. ]

        * If consensus-method 25 or later is in use, then we vote
          on recommended-protocols and required-protocols lines in the
          consensus.  We also include protocols lines in routerstatus
          entries.

        * If consensus-method 26 or later is in use, then we
initialize
          bandwidth weights to 1 in our calculations, to avoid
          division-by-zero errors on unusual networks.

        * If consensus method 27 or later is used, the microdesc
consensus
          may include an "a" line for each router, listing an IPv6 OR
port.

        [ As of 0.4.3.1-alpha, authorities no longer advertise or
negotiate
          any consensus methods lower than 28. ]

        * If consensus method 28 or later is used, microdescriptors no
longer
          include "a" lines.

        * If consensus method 29 or later is used, microdescriptor
"family"
          lines are canonicalized to improve compression.

        * If consensus method 30 or later is used, the base64 encoded
          ntor-onion-key does not include the trailing = sign.

        * If consensus method 31 or later is used, authorities parse
the
          "bwweightscale" and "maxunmeasuredbw" parameters correctly
when
          computing votes.

        * If consensus method 32 or later is used, authorities handle
the
          "MiddleOnly" flag specially when computing a consensus.
When the
          voters agree to include "MiddleOnly" in a routerstatus, they
          automatically remove "Exit", "Guard", "V2Dir", and "HSDir".
If
          the BadExit flag is included in the consensus, they
automatically
          add it to the routerstatus.

        * If consensus method 33 or later is used, and the consensus
          flavor is "microdesc", then the "Publication" field in the
"r"
          line is set to "2038-01-01 00:00:00".
```

```
        * If consensus method 34 or later is used, the consensus
          does not include any "package" lines.
```

The signatures at the end of a consensus document are sorted in ascending order by identity digest.

All ties in computing medians are broken in favor of the smaller or earlier item.

Deciding which Ids to include

This sorting algorithm is used for consensus-method 22 and later.

```
  First, consider each listing by tuple of <Ed,Rsa> identities, where
'Ed'
    may be "None" if the voter included "id ed25519 none" to indicate
that
    the authority knows what ed25519 identities are, and thinks that
the RSA
    key doesn't have one.

  For each such <Ed, RSA> tuple that is listed by more than half of
the
    total authorities (not just total votes), include it.  (It is not
    possible for any other <id-Ed, id-RSA'> to have as many votes.)
If more
    than half of the authorities list a single <Ed,Rsa> pair of this
type, we
    consider that Ed key to be "consensus"; see description of the
    NoEdConsensus flag.

  Log any other id-RSA values corresponding to an id-Ed we included,
and any
    other id-Ed values corresponding to an id-RSA we included.

  For each <id-RSA> that is not yet included, if it is listed by more
than
    half of the total authorities, and we do not already have it
listed with
    some <id-Ed>, include it, but do not consider its Ed identity
canonical.
```

Deciding which descriptors to include

Deciding which descriptors to include.

General-use HTTP URLs

Unless otherwise stated, “fingerprints” in these URLs are base16-encoded SHA1 hashes.

> Note that these URLs have variants ending in “.z”.

The most recent v3 consensus should be available at:

`http://<hostname>/tor/status-vote/current/consensus`

Similarly, the v3 microdescriptor consensus should be available at:

`http://<hostname>/tor/status-vote/current/consensus-microdesc`

A directory cache SHOULD start serving a new consensus document as soon as it is available and valid, regardless of how many router microdescriptors the cache is missing.

> Clients with an existing valid copy of the Tor directory won’t be troubled by these missing descriptors because they don’t start to use a new consensus right away.
>
> Even for new clients, in practice most microdescriptors don’t change from one consensus to the next, so a cache is likely to have a complete enough set even if it serves a new consensus right away.

Starting with Tor version 0.2.1.1-alpha is also available at:

`http://<hostname>/tor/status-vote/current/consensus/<F1>+<F2>+<F3>`

Where F1, F2, etc. are fingerprints of the authority identity keys (`SHA1(DER(KP_auth_id_rsa))`) for the authorities that the client trusts. Servers will only return a consensus if more than half of the requested authorities have signed the document, otherwise a 404 error will be sent back. The fingerprints can be shortened to a length of any multiple of two, using only the leftmost part of the encoded fingerprint. Tor uses 3 bytes (6 hex characters) of the fingerprint.

Clients SHOULD sort the fingerprints in ascending order. Server MUST accept any order.

Consensus-negotiation timeline

```
Period begins: this is the Published time.
  Everybody sends votes
Reconciliation: everybody tries to fetch missing votes.
  consensus may exist at this point.
End of voting period:
  everyone swaps signatures.
Now it's okay for caches to download
  Now it's okay for clients to download.

Valid-after/valid-until switchover
```

A tuple belongs to an `<id-RSA, id-Ed>` identity if it is a new tuple that matches both ID parts, or if it is an old tuple (one with no Ed opinion) that matches the RSA part. A tuple belongs to an `<id-RSA>` identity if its RSA identity matches.

A tuple matches another tuple if all the fields that are present in both tuples are the same.

For every included identity, consider the tuples belonging to that identity. Group them into sets of matching tuples. Include the tuple that matches the largest set, breaking ties in favor of the most recently published, and then in favor of the smaller server descriptor digest.

Forward compatibility

Future versions of Tor will need to include new information in the consensus documents, but it is important that all authorities (or at least half) generate and sign the same signed consensus.

To achieve this, authorities list in their votes their supported methods for generating consensuses from votes. Later methods will be assigned higher numbers. Currently specified methods:

```
     "1" -- The first implemented version.
     "2" -- Added support for the Unnamed flag.
     "3" -- Added legacy ID key support to aid in authority ID key
rollovers
     "4" -- No longer list routers that are not running in the
consensus
     "5" -- adds support for "w" and "p" lines.
     "6" -- Prefers measured bandwidth values rather than advertised
     "7" -- Provides keyword=integer pairs of consensus parameters
     "8" -- Provides microdescriptor summaries
     "9" -- Provides weights for selecting flagged routers in paths
     "10" -- Fixes edge case bugs in router flag selection weights
     "11" -- Don't consider BadExits when calculating bandwidth
weights
     "12" -- Params are only included if enough auths voted for them
     "13" -- Omit router entries with missing microdescriptors.
     "14" -- Adds support for "a" lines in ns consensuses and
microdescriptors.
     "15" -- Adds support for "p6" lines.
     "16" -- Adds ntor keys to microdescriptors
     "17" -- Adds "Unmeasured=1" flags to "w" lines
     "18" -- Adds 'id' to microdescriptors.
     "19" -- Adds "package" lines to consensuses
     "20" -- Adds GuardFraction information to microdescriptors.
     "21" -- Adds Ed25519 keys to microdescriptors.
     "22" -- Instantiates Ed25519 voting algorithm correctly.
     "23" -- Adds shared randomness protocol data.
     "24" -- No longer lists routers that are not Valid in the
consensus.
     "25" -- Vote on recommended-protocols and required-protocols.
     "26" -- Initialize bandwidth weights to 1 to avoid division-by-
zero.
     "27" -- Adds support for "a" lines in microdescriptor
consensuses.
     "28" -- Removes "a" lines from microdescriptors.
     "29" -- Canonicalizes families in microdescriptors.
     "30" -- Removes padding from ntor-onion-key.
     "31" -- Uses correct parsing for bwweightscale and
maxunmeasuredbw
             when computing weights
     "32" -- Adds special handling for MiddleOnly flag.
     "33" -- Sets "publication" field in microdesc consensus "r" lines
             to a meaningless value.
     "34" -- Removes "package" lines from consensus.
     "35" -- Includes "family-ids" entry in microdescriptors.
```

Before generating a consensus, an authority must decide which consensus method to use. To do this, it looks for the highest version number supported by more than 2/3 of the authorities voting. If it supports this method, then it uses it. Otherwise, it falls back to the newest consensus method that it supports (which will probably not result in a sufficiently signed consensus).

```
  200 -- the operation completed successfully
      -- the user requested statuses or serverdescs, and none of the
ones we
         requested were found (0.2.0.4-alpha and earlier).

  304 -- the client specified an if-modified-since time, and none of
the
         requested resources have changed since that time.

  400 -- the request is malformed, or
      -- the URL is for a malformed variation of one of the URLs we
support,
          or
      -- the client tried to post to a non-authority, or
      -- the authority rejected a malformed posted document, or

  404 -- the requested document was not found.
      -- the user requested statuses or serverdescs, and none of the
ones
         requested were found (0.2.0.5-alpha and later).

  503 -- we are declining the request in order to save bandwidth
      -- user requested some items that we ordinarily generate or
store,
         but we do not have any available.
```

> compressions: `deflate` and `x-zstd` compression are cheap enough that it can be calculated on-the-fly in response to each directory client request.

Note that for anonymous directory requests (that is, requests made over multi-hop circuits, like those for onion service lookups) implementations SHOULD NOT advertise any Accept-Encoding values other than deflate. To do so would be to create a fingerprinting opportunity.

When receiving multiple documents, clients MUST accept compressed concatenated documents and concatenated compressed documents as equivalent.

Servers MAY set the Content-Length: header. When they do, it should match the number of compressed bytes that they are sending.

Servers MAY include an X-Your-Address-Is: header, whose value is the apparent IP address of the client connecting to them (as a dotted quad). For directory connections tunneled over a BEGIN_DIR stream, servers SHOULD report the IP from which the circuit carrying the BEGIN_DIR stream reached them.

Servers SHOULD disable caching of multiple network statuses or multiple server descriptors. Servers MAY enable caching of single descriptors, single network statuses, the list of all server descriptors, a v1 directory, or a v1 running routers document. XXX mention times.

HTTP status codes

Tor delivers the following status codes. Some were chosen without much thought; other code SHOULD NOT rely on specific status codes yet.

All authorities MUST support method 25; authorities SHOULD support more recent methods as well. Authorities SHOULD NOT support or advertise support for any method before 25. Clients MAY assume that they will never see a current valid signed consensus for any method before method 25.

(The consensuses generated by new methods must be parsable by implementations that only understand the old methods, and must not cause those implementations to compromise their anonymity. This is a means for making changes in the contents of consensus; not for making backward-incompatible changes in their format.)

The following methods have incorrect implementations; authorities SHOULD NOT advertise support for them:

“21” – Did not correctly enable support for ed25519 key collation.

Exit policy summaries

Exit policy summaries appear in:

- `ipv6-policy` in server descriptors
- `p` in votes and consensuses

The format is:

- *keyword* `accept`|`reject` *PortList*
- ***PortList*** = *_PortList* **,** *PortOrRange*
- ***PortOrRange*** = INT – INT / INT

> For the avoidance of doubt, *PortList* is a single argument, so it cannot contain spaces.

Whether the summary shows the list of accepted ports or the list of rejected ports depends on which list is shorter (has a shorter string representation). In case of ties we choose the list of accepted ports. As an exception to this rule an allow-all policy is represented as “accept 1-65535” instead of “reject “ and a reject-all policy is similarly given as “reject 1-65535”.

Summary items are compressed, that is instead of “80-88,89-100” there only is a single item of “80-100”, similarly instead of “20,21” a summary will say “20-21”.

Port lists are sorted in ascending order.

The maximum allowed length of a policy summary (including the "accept " or "reject ") is 1000 characters. If a summary exceeds that length we use an accept-style summary and list as much of the port list as is possible within these 1000 bytes. [XXXX be more specific.]

Computing Bandwidth Weights

Let weight_scale = 10000, or the value of the "bwweightscale" parameter. (Before consensus method 31 there was a bug in parsing bwweightscale, so that if there were any consensus parameters after it alphabetically, it would always be treated as 10000. A similar bug existed for "maxunmeasuredbw".)

Starting with consensus method 26, G, M, E, and D are initialized to 1 and T to 4. Prior consensus methods initialize them all to 0. With this change, test tor networks that are small or new are much more likely to produce bandwidth-weights in their consensus. The extra bandwidth has a negligible impact on the bandwidth weights in the public tor network.

Let G be the total bandwidth for Guard-flagged nodes. Let M be the total bandwidth for non-flagged nodes. Let E be the total bandwidth for Exit-flagged nodes. Let D be the total bandwidth for Guard+Exit-flagged nodes. Let T = G+M+E+D

Let Wgd be the weight for choosing a Guard+Exit for the guard position. Let Wmd be the weight for choosing a Guard+Exit for the middle position. Let Wed be the weight for choosing a Guard+Exit for the exit position.

Let Wme be the weight for choosing an Exit for the middle position. Let Wmg be the weight for choosing a Guard for the middle position.

Let Wgg be the weight for choosing a Guard for the guard position. Let Wee be the weight for choosing an Exit for the exit position.

Balanced network conditions then arise from solutions to the following system of equations:

Wgg*G* + *Wgd*D == M + Wmd*D* + *Wme*E + Wmg*G (guard bw = middle bw) Wgg*G + Wgd*D == *Wee*E + Wed*D (guard bw = exit bw) Wed*D + Wmd*D* + *Wgd*D == D (aka: Wed+Wmd+Wdg = weight_scale) Wmg*G* + *Wgg*G == G (aka: Wgg = weight_scale-Wmg) Wme*E* + *Wee*E == E (aka: Wee = weight_scale-Wme)

We are short 2 constraints with the above set. The remaining constraints come from examining different cases of network load. The following constraints are

> Besides, the entire concept of this very advanced header syntax might have looked promising in the late 90s, but it is a rarely used feature these days, leading to not much of a good reason to support it.

To support older clients, and obsolete software, directory servers MUST also support GET requests to URLs with a ".z" suffix appended.

The semantics are as follows:

- Clients SHOULD NOT request the `.z` URL.
- Clients MUST NOT request a `.z` URL and include an `Accept-Encoding` hesder that fails to advertise `deflate`.
- If the client does not send an `Accept-Encoding` header along with a `.z` URL, the server MUST send the response compressed with `deflate` and SHOULD NOT send a `Content-Encoding` header.
- If the client *does* send an `Accept-Encoding` header along with a `.z` URL, the server SHOULD treat the request the same way as for the URL without the `.z`. If `deflate` is included in the `Accept-Encoding`, the response MUST be encoded, once, with an encoding advertised by the client, and be accompanied by an appropriate `Content-Encoding`.

Note that these semantics are irreconcilable with the HTTP specifications, and may give rise to malfunctions or inconsistencies when `.z` URLs are queried by normal, standards-conforming, HTTP clients. This suffix is allowed on *all* HTTP GET request URLs, except as explicitly noted. It is not supported on any HTTP POST request URLs.

> Tor clients started sending `Accept-Encoding` in 0.3.1.1-alpha, but they still request `.z` URLs when sending `Accept-Encoding`. Up until June 2025, Arti requested `.z` URLs. Other software that sends `.z` URLs probably also exists.

For all documents, servers MUST support `identity` and `deflate`, and SHOULD support `x-zstd`. Servers SHOULD support serving `current/consensus` and `current/consensus-microdesc` with `x-tor-lzma` compression; this includes consensus diffs.

For all documents, clients MUST support `identity` and `deflate`.

> For performance reasons, it will usually be necessary for each directory server to precalculate or cache the `x-tor-lzma` compressed form of the `consensus*` documents and diffs. For other documents and other

Standards compliance

All clients and servers MUST support HTTP 1.0. Clients and servers MAY support later versions of HTTP as well.

HTTP headers

Servers SHOULD set Content-Encoding to the algorithm used to compress the document(s) being served. Recognized algorithms are:

```
      - "identity"     -- RFC2616 section 3.5
      - "deflate"      -- RFC2616 section 3.5
      - "gzip"         -- RFC2616 section 3.5
      - "x-zstd"       -- The zstandard compression algorithm
 (www.zstd.net)
      - "x-tor-lzma"   -- The lzma compression algorithm, with a
 "preset"
                          value no higher than 6.
```

Clients SHOULD use Accept-Encoding on most directory requests to indicate which of the above compression algorithms they support.

Clients MUST NOT send wildcards (`*`) or qvalue weightings (`<ALGORITHM>;q=<0-1>`).

Clients SHOULD NOT assume that the order of the supported algorithms in Accept-Encoding carries any signifcance.

Clients SHOULD write the Accept-Encoding as a single line separated by `, `.

In general, the grammar for the Accept-Encoding header can be summarized as:

```
<ALGORITHM> ::= "identity" | "deflate" | "gzip" | "x-zstd" | "x-tor-
lzma"
<ACCEPT_ENCODING_VALUE> ::= <ALGORITHM> [ ", ", <ALGORITHM> ]*
```

> The reason for these limitations lie within both: the primitiveness of the ctor `parse_accept_encoding_header` function as well as the lack of a widespread Rust library to support spec compliant parsing of this header, requiring us to implement it ourselves.

used in consensus method 10 and above. There are another incorrect and obsolete set of constraints used for these same cases in consensus method 9. For those, see dir-spec.txt in Tor 0.2.2.10-alpha to 0.2.2.16-alpha.

Case 1: E >= T/3 && G >= T/3 (Neither Exit nor Guard Scarce)

In this case, the additional two constraints are: Wmg == Wmd, Wed == 1/3.

```
     This leads to the solution:
         Wgd = weight_scale/3
         Wed = weight_scale/3
         Wmd = weight_scale/3
         Wee = (weight_scale*(E+G+M))/(3*E)
         Wme = weight_scale - Wee
         Wmg = (weight_scale*(2*G-E-M))/(3*G)
         Wgg = weight_scale - Wmg

   Case 2: E < T/3 && G < T/3 (Both are scarce)
```

Let R denote the more scarce class (Rare) between Guard vs Exit. Let S denote the less scarce class.

Subcase a: R+D < S

In this subcase, we simply devote all of D bandwidth to the scarce class.

```
      Wgg = Wee = weight_scale
      Wmg = Wme = Wmd = 0;
      if E < G:
        Wed = weight_scale
        Wgd = 0
      else:
        Wed = 0
        Wgd = weight_scale

   Subcase b: R+D >= S
```

In this case, if M <= T/3, we have enough bandwidth to try to achieve a balancing condition.

Add constraints Wgg = weight_scale, Wmd == Wgd to maximize bandwidth in the guard position while still allowing exits to be used as middle nodes:

Wee = (weight_scale*(E - G + M))/E Wed = (weight_scale*(D - 2*E* + *4*G - 2*M))/(3*D) Wme = (weight_scale*(G-M))/E Wmg = 0 Wgg = weight_scale Wmd = (weight_scale - Wed)/2 Wgd = (weight_scale - Wed)/2

If this system ends up with any values out of range (ie negative, or above weight_scale), use the constraints Wgg == weight_scale and Wee == weight_scale, since both those positions are scarce:

```
   Consensus objects, as a non-bridge cache:
         0 (TestingServerConsensusDownloadInitialDelay)

   Consensus objects, as a client or bridge that has bootstrapped:
         0 (TestingClientConsensusDownloadInitialDelay)

   Consensus objects, as a client or bridge that is bootstrapping,
   when connecting to an authority because no "fallback" caches are
   known:
         0 (ClientBootstrapConsensusAuthorityOnlyDownloadInitialDelay)

   Consensus objects, as a client or bridge that is bootstrapping,
   when "fallback" caches are known but connecting to an authority
   anyway:
         6 (ClientBootstrapConsensusAuthorityDownloadInitialDelay)

   Consensus objects, as a client or bridge that is bootstrapping,
   when downloading from a "fallback" cache.
         0 (ClientBootstrapConsensusFallbackDownloadInitialDelay)

   Bridge descriptors, as a bridge-using client when at least one
bridge
   is usable:
         10800 (TestingBridgeDownloadInitialDelay)

   Bridge descriptors, otherwise:
         0 (TestingBridgeBootstrapDownloadInitialDelay)

   Other objects, as cache or authority:
         0 (TestingServerDownloadInitialDelay)

   Other objects, as client:
         0 (TestingClientDownloadInitialDelay)
```

download. To determine the amount of time to wait, clients use a randomized exponential backoff algorithm. (Specifically, they use a variation of the "decorrelated jitter" algorithm from https://aws.amazon.com/blogs/architecture/exponential-backoff-and-jitter/ .)

The specific formula used to compute the 'i+1'th delay is:

```
        Delay_0     = 0

        Delay_{i+1} = MIN(cap, random_between(lower_bound,
upper_bound)))
          where upper_bound = MAX(lower_bound + epsilon, Delay_i * 3)
                lower_bound = MAX(1, base_delay).
```

After the first download attempt fails, we wait for `Delay_1` before retrying; after the second failed attempt, we wait for `Delay_2`, and so on.

The value of `cap` is whatever largest duration we can conveniently represent (such as `INT_MAX` seconds, or `U32_MAX` milliseconds); the value of `epsilon` is the unit of time we're using for our calculations (typically 1 second or 1 millisecond); the value of `base_delay` depends on what is being downloaded, whether the client is fully bootstrapped, how the client is configured, and where it is downloading from. Current base_delay values are:

```
        Wgg = weight_scale
        Wee = weight_scale
        Wed = (weight_scale*(D - 2*E + G + M))/(3*D)
        Wmd = (weight_Scale*(D - 2*M + G + E))/(3*D)
        Wme = 0
        Wmg = 0
        Wgd = weight_scale - Wed - Wmd

    If M > T/3, then the Wmd weight above will become negative. Set
it to 0
    in this case:
       Wmd = 0
       Wgd = weight_scale - Wed

  Case 3: One of E < T/3 or G < T/3

    Let S be the scarce class (of E or G).

    Subcase a: (S+D) < T/3:
      if G=S:
        Wgg = Wgd = weight_scale;
        Wmd = Wed = Wmg = 0;
        // Minor subcase, if E is more scarce than M,
        // keep its bandwidth in place.
        if (E < M) Wme = 0;
        else Wme = (weight_scale*(E-M))/(2*E);
        Wee = weight_scale-Wme;
      if E=S:
        Wee = Wed = weight_scale;
        Wmd = Wgd = Wme = 0;
        // Minor subcase, if G is more scarce than M,
        // keep its bandwidth in place.
        if (G < M) Wmg = 0;
        else Wmg = (weight_scale*(G-M))/(2*G);
        Wgg = weight_scale-Wmg;

    Subcase b: (S+D) >= T/3
      if G=S:
        Add constraints Wgg = weight_scale, Wmd == Wed to maximize
bandwidth
        in the guard position, while still allowing exits to be
        used as middle nodes:
          Wgg = weight_scale
          Wgd = (weight_scale*(D - 2*G + E + M))/(3*D)
          Wmg = 0
          Wee = (weight_scale*(E+M))/(2*E)
          Wme = weight_scale - Wee
          Wmd = (weight_scale - Wgd)/2
          Wed = (weight_scale - Wgd)/2
      if E=S:
        Add constraints Wee == weight_scale, Wmd == Wgd to maximize
bandwidth
        in the exit position:
          Wee = weight_scale;
```

```
Wed = (weight_scale*(D - 2*E + G + M))/(3*D);
Wme = 0;
Wgg = (weight_scale*(G+M))/(2*G);
Wmg = weight_scale - Wgg;
Wmd = (weight_scale - Wed)/2;
Wgd = (weight_scale - Wed)/2;
```

To ensure consensus, all calculations are performed using integer math with a fixed precision determined by the bwweightscale consensus parameter (defaults at 10000, Min: 1, Max: INT32_MAX). (See note above about parsing bug in bwweightscale before consensus method 31.)

For future balancing improvements, Tor clients support 11 additional weights for directory requests and middle weighting. These weights are currently set at weight_scale, with the exception of the following groups of assignments:

Directory requests use middle weights:

Wbd=Wmd, Wbg=Wmg, Wbe=Wme, Wbm=Wmm

Handle bridges and strange exit policies:

Wgm=Wgg, Wem=Wee, Weg=Wed

Computing consensus flavors

Consensus flavors are variants of the consensus that clients can choose to download and use instead of the unflavored consensus. The purpose of a consensus flavor is to remove or replace information in the unflavored consensus without forcing clients to download information they would not use anyway.

Directory authorities can produce and serve an arbitrary number of flavors of the same consensus. A downside of creating too many new flavors is that clients will be distinguishable based on which flavor they download. A new flavor should not be created when adding a field instead wouldn't be too onerous.

Examples for consensus flavors include:

```
- Publishing hashes of microdescriptors instead of hashes of
  full descriptors (see section 3.9.2).
- Including different digests of descriptors, instead of the
  perhaps-soon-to-be-totally-broken SHA1.
```

Consensus flavors are derived from the unflavored consensus once the voting process is complete. This is to avoid consensus synchronization problems.

the third, 10 minutes for the fourth, and 1 day thereafter.) Periodically (currently once an hour) clients reset the failure count.

Clients retain the most recent descriptor they have downloaded for each router so long as it is listed in the consensus. If it is not listed, they keep it so long as it is not too old (currently, ROUTER_MAX_AGE=48 hours) and no better router descriptor has been downloaded for the same relay. Caches retain descriptors until they are at least OLD_ROUTER_DESC_MAX_AGE=5 days old.

Clients which chose to download the microdescriptor consensus instead of the general consensus must download the referenced microdescriptors instead of server descriptors. Clients fetch and cache microdescriptors preemptively from dir mirrors when starting up, like they currently fetch descriptors. After bootstrapping, clients only need to fetch the microdescriptors that have changed.

When a client gets a new microdescriptor consensus, it looks to see if there are any microdescriptors it needs to learn, and launches a request for them.

Clients maintain a cache of microdescriptors along with metadata like when it was last referenced by a consensus, and which identity key it corresponds to. They keep a microdescriptor until it hasn't been mentioned in any consensus for a week. Future clients might cache them for longer or shorter times.

Downloading extra-info documents

Any client that uses extra-info documents should implement this section.

Note that generally, clients don't need extra-info documents.

Periodically, the Tor instance checks whether it is missing any extra-info documents: in other words, if it has any server descriptors with an extra-info-digest field that does not match any of the extra-info documents currently held. If so, it downloads whatever extra-info documents are missing. Clients try to download from caches. We follow the same splitting and back-off rules as in section 5.2.

Retrying failed downloads

This section applies to caches as well as to clients.

When a client fails to download a resource (a consensus, a router descriptor, a microdescriptor, etc) it waits for a certain amount of time before retrying the

Downloading server descriptors or microdescriptors

Clients try to have the best descriptor for each router. A descriptor is “best” if:

- It is listed in the consensus network-status document.

Periodically (currently every 10 seconds) clients check whether there are any “downloadable” descriptors. A descriptor is downloadable if:

```
      - It is the "best" descriptor for some router.
      - The descriptor was published at least 10 minutes in the past.
        (This prevents clients from trying to fetch descriptors that
the
        mirrors have probably not yet retrieved and cached.)
      - The client does not currently have it.
      - The client is not currently trying to download it.
      - The client would not discard it immediately upon receiving it.
      - The client thinks it is running and valid (see section 5.4.1
below).
```

If at least 16 known routers have downloadable descriptors, or if enough time (currently 10 minutes) has passed since the last time the client tried to download descriptors, it launches requests for all downloadable descriptors.

When downloading multiple server descriptors, the client chooses multiple mirrors so that:

```
     - At least 3 different mirrors are used, except when this would
result
       in more than one request for under 4 descriptors.
     - No more than 128 descriptors are requested from a single
mirror.
     - Otherwise, as few mirrors as possible are used.
   After choosing mirrors, the client divides the descriptors among
them
   randomly.
```

After receiving any response the client MUST discard any descriptors that it did not request.

When a descriptor download fails, the client notes it, and does not consider the descriptor downloadable again until a certain amount of time has passed. (Currently 0 seconds for the first failure, 60 seconds for the second, 5 minutes for

Every consensus flavor has a name consisting of a sequence of one or more alphanumeric characters and dashes. For compatibility, the original (unflavored) consensus type is called “ns”.

The supported consensus flavors are defined as part of the authorities’ consensus method.

All consensus flavors have in common that their first line is “network-status-version” where version is 3 or higher, and the flavor is a string consisting of alphanumeric characters and dashes:

“network-status-version” SP version [SP flavor] NL

ns consensus

The ns consensus flavor is equivalent to the unflavored consensus. When the flavor is omitted from the “network-status-version” line, it should be assumed to be “ns”. Some implementations may explicitly state that the flavor is “ns” when generating consensuses, but should accept consensuses where the flavor is omitted.

Microdescriptor consensus

The microdescriptor consensus is a consensus flavor that contains microdescriptor hashes instead of descriptor hashes and that omits exit-policy summaries which are contained in microdescriptors. The microdescriptor consensus was designed to contain elements that are small and frequently changing. Clients use the information in the microdescriptor consensus to decide which servers to fetch information about and which servers to fetch information from.

The microdescriptor consensus is based on the unflavored consensus with the exceptions as follows:

“network-status-version” SP version SP “microdesc” NL

[At start, exactly once.]

The flavor name of a microdescriptor consensus is “microdesc”.

Changes to router status entries are as follows:

```
    "r" SP nickname SP identity SP publication SP IP SP ORPort
        SP DirPort NL

        [At start, exactly once.]

        Similar to "r" lines in section 3.4.1, but without the digest
element.

    "a" SP address ":" port NL

        [Any number]

        Identical to the "a" lines in section 3.4.1.
```

(Only included when the vote is generated with consensus-method 14 or later, and the consensus is generated with consensus-method 27 or later.)

“p” ... NL

[At most once]

Not currently generated.

Exit policy summaries are contained in microdescriptors and therefore omitted in the microdescriptor consensus.

“m” SP digest NL

[Exactly once.*]

“digest” is the base64 of the SHA256 hash of the router’s microdescriptor with trailing =s omitted. For a given router descriptor digest and consensus method there should only be a single microdescriptor digest in the “m” lines of all votes. If different votes have different microdescriptor digests for the same descriptor digest and consensus method, at least one of the authorities is broken. If this happens, the microdesc consensus should contain whichever microdescriptor digest is most common. If there is no winner, we break ties in the favor of the lexically earliest.

[*Before consensus method 13, this field was sometimes erroneously omitted.]

Additionally, a microdescriptor consensus SHOULD use the sha256 digest algorithm for its signatures.

```
  [Newer versions of Tor (0.2.6.2-alpha and later):
   If the consensus contains Exits (the typical case), Tor will build
both
   exit and internal circuits. When bootstrap completes, Tor will be
ready
   to handle an application requesting an exit circuit to services
like the
   World Wide Web.
```

If the consensus does not contain Exits, Tor will only build internal circuits. In this case, earlier statuses will have included “internal” as indicated above. When bootstrap completes, Tor will be ready to handle an application requesting an internal circuit to hidden services at “.onion” addresses.

If a future consensus contains Exits, exit circuits may become available.]

(Note: clients can and should pick caches based on the network-status information they have: once they have first fetched network-status info from an authority or fallback, they should not need to go to the authority directly again, and should only choose the fallback at random, based on its consensus weight in the current consensus.)

To avoid swarming the caches whenever a consensus expires, the clients download new consensuses at a randomly chosen time after the caches are expected to have a fresh consensus, but before their consensus will expire. (This time is chosen uniformly at random from the interval between the time 3/4 into the first interval after the consensus is no longer fresh, and 7/8 of the time remaining after that before the consensus is invalid.)

```
   [For example, if a client has a consensus that became valid at
1:00,
    and is fresh until 2:00, and expires at 4:00, that client will
fetch
    a new consensus at a random time between 2:45 and 3:50, since 3/4
    of the one-hour interval is 45 minutes, and 7/8 of the remaining
75
    minutes is 65 minutes.]
```

Clients may choose to download the microdescriptor consensus instead of the general network status consensus. In that case they should use the same update strategy as for the normal consensus. They should not download more than one consensus flavor.

When a client does not have a live consensus, it will generally use the most recent consensus it has if that consensus is “reasonably live”. A “reasonably live” consensus is one that expired less than 24 hours ago.

Client operation

Every Tor that is not a directory server (that is, those that do not have a DirPort set) implements this section.

Downloading network-status documents

Each client maintains a list of directory authorities. Insofar as possible, clients SHOULD all use the same list.

```
   [Newer versions of Tor (0.2.8.1-alpha and later):
    Each client also maintains a list of default fallback directory
mirrors
   (fallbacks). Each released version of Tor MAY have a different
list,
   depending on the mirrors that satisfy the fallback directory
criteria at
   release time.]
```

Clients try to have a live consensus network-status document at all times. A network-status document is "live" if the time in its valid-after field has passed, and the time in its valid-until field has not passed.

When a client has no consensus network-status document, it downloads it from a randomly chosen fallback directory mirror or authority. Clients prefer fallbacks to authorities, trying them earlier and more frequently. In all other cases, the client downloads from caches randomly chosen from among those believed to be V3 directory servers. (This information comes from the network-status documents.)

After receiving any response client MUST discard any network-status documents that it did not request.

On failure, the client waits briefly, then tries that network-status document again from another cache. The client does not build circuits until it has a live network-status consensus document, and it has descriptors for a significant proportion of the routers that it believes are running (this is configurable using torrc options and consensus parameters).

Exchanging detached signatures

> Note that many of the URLs here have variants ending in ".z".

Once an authority has computed and signed a consensus network status, it should send its detached signature to each other authority in an HTTP POST request to the URL:

`http://<hostname>/tor/post/consensus-signature`

[XXX Note why we support push-and-then-pull.]

> Note that since this is a POST request, the legacy ".z" suffix is not supported.

All of the detached signatures it knows for consensus status should be available at:

`http://<hostname>/tor/status-vote/next/consensus-signatures`

Assuming full connectivity, every authority should compute and sign the same consensus including any flavors in each period. Therefore, it isn't necessary to download the consensus or any flavors of it computed by each authority; instead, the authorities only push/fetch each others' signatures. A "detached signature" document contains items as follows:

"consensus-digest" SP Digest NL

[At start, at most once.]

The digest of the consensus being signed.

"valid-after" SP YYYY-MM-DD SP HH:MM:SS NL "fresh-until" SP YYYY-MM-DD SP HH:MM:SS NL "valid-until" SP YYYY-MM-DD SP HH:MM:SS NL

[As in the consensus]

"additional-digest" SP flavor SP algname SP digest NL

[Any number.]

For each supported consensus flavor, every directory authority adds one or more "additional-digest" lines. "flavor" is the name of the consensus flavor, "algname" is the name of the hash algorithm that is used to generate the digest, and "digest" is the hex-encoded digest.

The hash algorithm for the microdescriptor consensus flavor is defined as SHA256 with algname “sha256”.

```
"additional-signature" SP flavor SP algname SP identity SP
     signing-key-digest NL signature.

    [Any number.]
```

For each supported consensus flavor and defined digest algorithm, every directory authority adds an “additional-signature” line. “flavor” is the name of the consensus flavor. “algname” is the name of the algorithm that was used to hash the identity and signing keys, and to compute the signature. “identity” is the hex-encoded digest of the authority identity key of the signing authority, and “signing-key-digest” is the hex-encoded digest of the current authority signing key of the signing authority.

The “sha256” signature format is defined as the RSA signature of the OAEP+-padded SHA256 digest of the item to be signed. When checking signatures, the signature MUST be treated as valid if the signature material begins with SHA256(document), so that other data can get added later. [To be honest, I didn’t fully understand the previous paragraph and only copied it from the proposals. Review carefully. -KL]

“directory-signature”

[As in the consensus; the signature object is the same as in the consensus document.]

Serving and requesting diffs

When downloading the current consensus, a client may include an HTTP header of the form

X-Or-Diff-From-Consensus: HASH1, HASH2, ...

where the HASH values are hex-encoded SHA3-256 digests of the *signed part* of one or more consensuses that the client knows about.

If a cache knows a consensus diff from one of those consensuses to the most recent consensus of the requested flavor, it may send that diff instead of the specified consensus.

Caches also serve diffs from the URIs:

```
/tor/status-vote/current/consensus/diff/<HASH>/<FPRLIST>
/tor/status-vote/current/consensus-<FLAVOR>/diff/<HASH>/<FPRLIST>
```

where FLAVOR is the consensus flavor, defaulting to “ns”, and FPRLIST is +-separated list of recognized authority identity fingerprints as in appendix B.

> Note that these URLs here have variants ending in “.z”.

Retrying failed downloads

See section 5.5 below; it applies to caches as well as clients.

Downloading extra-info documents from directory authorities

Any cache that chooses to cache extra-info documents should implement this section.

Periodically, the Tor instance checks whether it is missing any extra-info documents: in other words, if it has any server descriptors with an extra-info-digest field that does not match any of the extra-info documents currently held. If so, it downloads whatever extra-info documents are missing. Caches download from authorities. We follow the same splitting and back-off rules as in section 4.2.

Consensus diffs

Instead of downloading an entire consensus, clients may download a "diff" document containing an ed-style diff from a previous consensus document. Caches (and authorities) make these diffs as they learn about new consensuses. To do so, they must store a record of older consensuses.

Support for consensus diffs was added in 0.3.1.1-alpha, and is advertised with the subprotocol "DirCache=2" (`DIRCACHE_CONSDIFF`).

Consensus diff format

Consensus diffs are formatted as follows:

The first line is "network-status-diff-version 1" NL

The second line is

"hash" SP FromDigest SP ToDigest NL

where FromDigest is the hex-encoded SHA3-256 digest of the *signed part* of the consensus that the diff should be applied to, and ToDigest is the hex-encoded SHA3-256 digest of the *entire* consensus resulting from applying the diff. (See 3.4.1 for information on that part of a consensus is signed.)

The third and subsequent lines encode the diff from FromDigest to ToDigest in a limited subset of the ed diff format, as specified in appendix E.

Publishing the signed consensus

> Note that the URLs here have variants ending in ".z".

The voting period ends at the valid-after time. If the consensus has been signed by a majority of authorities, these documents are made available at

```
http://<hostname>/tor/status-vote/current/consensus
```

and

```
http://<hostname>/tor/status-vote/current/consensus-signatures
```

```
   [XXX current/consensus-signatures is not currently implemented, as
it
    is not used in the voting protocol.]

   [XXX possible future features include support for downloading old
    consensuses.]

   The other vote documents are analogously made available under

http://<hostname>/tor/status-vote/current/authority
http://<hostname>/tor/status-vote/current/<fp>
http://<hostname>/tor/status-vote/current/d/<d>
http://<hostname>/tor/status-vote/current/bandwidth
```

once the voting period ends, regardless of the number of signatures.

The authorities serve another consensus of each flavor "F" from the locations

```
/tor/status-vote/(current|next)/consensus-F. and
/tor/status-vote/(current|next)/consensus-F/<FP1>+....
```

The standard URLs for bandwidth list files first-appeared in Tor 0.3.5.

Directory cache operation

All directory caches implement this section, except as noted.

> Note: Directory caches are currently in the process of being renamed to "Directory Mirrors", in order to better reflect their purpose. You might encounter both terms in the wild.

Downloading consensus status documents from directory authorities

All directory caches try to keep a recent network-status consensus document to serve to clients. A cache ALWAYS downloads a network-status consensus if any of the following are true:

- The cache has no consensus document.
- The cache's consensus document is no longer valid.

Otherwise, the cache downloads a new consensus document at a randomly chosen time in the first half-interval after its current consensus stops being fresh. (This time is chosen at random to avoid swarming the authorities at the start of each period. The interval size is inferred from the difference between the valid-after time and the fresh-until time on the consensus.)

```
[For example, if a cache has a consensus that became valid at 1:00,
 and is fresh until 2:00, that cache will fetch a new consensus at
 a random time between 2:00 and 2:30.]
```

Directory caches also fetch consensus flavors from the authorities. Caches check the correctness of consensus flavors, but do not check anything about an unrecognized consensus document beyond its digest and length. Caches serve all consensus flavors from the same locations as the directory authorities.

Downloading server descriptors from directory authorities

Periodically (currently, every 10 seconds), directory caches check whether there are any specific descriptors that they do not have and that they are not currently trying to download. Caches identify these descriptors by hash in the recent network-status consensus documents.

If so, the directory cache launches requests to the authorities for these descriptors.

If one of these downloads fails, we do not try to download that descriptor from the authority that failed to serve it again unless we receive a newer network-status consensus that lists the same descriptor.

Directory caches must potentially cache multiple descriptors for each router. Caches must not discard any descriptor listed by any recent consensus. If there is enough space to store additional descriptors, caches SHOULD try to hold those which clients are likely to download the most. (Currently, this is judged based on the interval for which each descriptor seemed newest.)

[XXXX define recent]

Downloading microdescriptors from directory authorities

Directory mirrors should fetch, cache, and serve each microdescriptor from the authorities.

The microdescriptors with base64 SHA256 hashes `<D1>`, `<D2>`, `<D3>` are available at:

`http://<hostname>/tor/micro/d/<D1>-<D2>-<D3>`

`<Dn>` are base64 encoded with trailing =s omitted for size and for consistency with the microdescriptor consensus format. -s are used instead of +s to separate items, since the + character is used in base64 encoding.

Directory mirrors should check to make sure that the microdescriptors they're about to serve match the right hashes (either the hashes from the fetch URL or the hashes from the consensus, respectively).

(NOTE: Due to squid proxy url limitations at most 92 microdescriptor hashes can be retrieved in a single request.)

> Note that these URLs here have variants ending in ".z".