Tor directory protocol, version 3

0. Scope and preliminaries

This directory protocol is used by Tor version 0.2.0.x-alpha and later. See dir-spec-v1.txt for information on the protocol used up to the 0.1.0.x series, and dir-spec-v2.txt for information on the protocol used by the 0.1.1.x and 0.1.2.x series.

This document merges and supersedes the following proposals:

```
    101  Voting on the Tor Directory System
    103  Splitting identity key from regularly used signing key
    104  Long and Short Router Descriptors
```

XXX timeline
XXX fill in XXXXs

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

0.1. History

The earliest versions of Onion Routing shipped with a list of known routers and their keys. When the set of routers changed, users needed to fetch a new list.

The Version 1 Directory protocol

Early versions of Tor (0.0.2) introduced "Directory authorities": servers that served signed "directory" documents containing a list of signed "server descriptors", along with short summary of the status of each router. Thus, clients could get up-to-date information on the state of the network automatically, and be certain that the list they were getting was attested by a trusted directory authority.

Later versions (0.0.8) added directory caches, which download directories from the authorities and serve them to clients. Non-caches fetch from the caches in preference to fetching from the authorities, thus distributing bandwidth requirements.

Also added during the version 1 directory protocol were "router status" documents: short documents that listed only the up/down status of the routers on the network, rather than a complete list of all the descriptors. Clients and caches would fetch these documents far more frequently than they would fetch full directories.

The Version 2 Directory Protocol

During the Tor 0.1.1.x series, Tor revised its handling of directory documents in order to address two major problems:

* Directories had grown quite large (over 1MB), and most directory downloads consisted mainly of server descriptors that clients already had.

* Every directory authority was a trust bottleneck: if a single directory authority lied, it could make clients believe for a time an arbitrarily distorted view of the Tor network. (Clients trusted the most recent signed document they downloaded.) Thus, adding more authorities would make the system less secure, not more.

To address these, we extended the directory protocol so that authorities now published signed "network status" documents. Each network status listed, for every router in the network: a hash of its identity key, a hash of its most recent descriptor, and a summary of what the authority believed about its status. Clients would download the authorities' network status documents in turn, and believe statements about routers iff they were attested to by more than half of the authorities.

Instead of downloading all server descriptors at once, clients downloaded only the descriptors that they did not have. Descriptors were indexed by their digests, in order to prevent malicious caches from giving different versions of a server descriptor to different clients.

Routers began working harder to upload new descriptors only when their contents were substantially changed.

0.2. Goals of the version 3 protocol

Version 3 of the Tor directory protocol tries to solve the following issues:

* A great deal of bandwidth used to transmit server descriptors was used by two fields that are not actually used by Tor routers (namely read-history and write-history). We save about 60% by moving them into a separate document that most clients do not fetch or use.

* It was possible under certain perverse circumstances for clients to download an unusual set of network status documents, thus partitioning themselves from clients who have a more recent and/or typical set of documents. Even under the best of circumstances, clients were sensitive to the ages of the network status documents they downloaded. Therefore, instead of having the clients correlate multiple network status documents, we have the authorities collectively vote on a single consensus network status document.

* The most sensitive data in the entire network (the identity keys of the directory authorities) needed to be stored unencrypted so that the authorities can sign network-status documents on the fly. Now, the authorities' identity keys are stored offline, and used to certify medium-term signing keys that can be rotated.

0.3. Some Remaining questions

Things we could solve on a v3 timeframe:

The SHA-1 hash is showing its age. We should do something about our dependency on it. We could probably future-proof ourselves here in this revision, at least so far as documents from the authorities are concerned.

Too many things about the authorities are hardcoded by IP.

Perhaps we should start accepting longer identity keys for routers too.

Things to solve eventually:

Requiring every client to know about every router won't scale forever.

Requiring every directory cache to know every router won't scale forever.

1. Outline

There is a small set (say, around 5-10) of semi-trusted directory authorities. A default list of authorities is shipped with the Tor software. Users can change this list, but are encouraged not to do so, in order to avoid partitioning attacks.

Every authority has a very-secret, long-term "Authority Identity Key". This is stored encrypted and/or offline, and is used to sign "key certificate" documents. Every key certificate contains a medium-term (3-12 months) "authority signing key", that is used by the authority to sign other directory information. (Note that the authority identity key is distinct from the router identity key that the authority uses in its role as an ordinary router.)

Routers periodically upload signed "routers descriptors" to the directory authorities describing their keys, capabilities, and other information. Routers may also upload signed "extra info documents" containing information that is not required for the Tor protocol. Directory authorities serve server descriptors indexed by router identity, or by hash of the descriptor.

Routers may act as directory caches to reduce load on the directory authorities. They announce this in their descriptors.

Periodically, each directory authority generates a view of the current descriptors and status for known routers. They send a signed summary of this view (a "status vote") to the other authorities. The authorities compute the result of this vote, and sign a "consensus status" document containing the result of the vote.

Directory caches download, cache, and re-serve consensus documents.

Clients, directory caches, and directory authorities all use consensus documents to find out when their list of routers is out-of-date. (Directory authorities also use vote statuses.) If it is, they download any missing server descriptors. Clients download missing descriptors from caches; caches and authorities download from authorities. Descriptors are downloaded by the hash of the descriptor, not by the relay's identity key: this prevents directory servers from attacking clients by giving them descriptors nobody else uses.

All directory information is uploaded and downloaded with HTTP.

1.1. What's different from version 2?

Clients used to download multiple network status documents, corresponding roughly to "status votes" above. They would compute the result of the vote on the client side.

Authorities used to sign documents using the same private keys they used
for their roles as routers. This forced them to keep these extremely
sensitive keys in memory unencrypted.

All of the information in extra-info documents used to be kept in the
main descriptors.

1.2. Document meta-format

Server descriptors, directories, and running-routers documents all obey the
following lightweight extensible information format.

The highest level object is a Document, which consists of one or more
Items. Every Item begins with a KeywordLine, followed by zero or more
Objects. A KeywordLine begins with a Keyword, optionally followed by
whitespace and more non-newline characters, and ends with a newline. A
Keyword is a sequence of one or more characters in the set [A-Za-z0-9-].
An Object is a block of encoded data in pseudo-Open-PGP-style
armor. (cf. RFC 2440)

More formally:

```
  NL = The ascii LF character (hex value 0x0a).
  Document ::= (Item | NL)+
  Item ::= KeywordLine Object*
  KeywordLine ::= Keyword NL | Keyword WS ArgumentChar+ NL
  Keyword = KeywordChar+
  KeywordChar ::= 'A' ... 'Z' | 'a' ... 'z' | '0' ... '9' | '-'
  ArgumentChar ::= any printing ASCII character except NL.
  WS = (SP | TAB)+
  Object ::= BeginLine Base64-encoded-data EndLine
  BeginLine ::= "-----BEGIN " Keyword "-----" NL
  EndLine ::= "-----END " Keyword "-----" NL

  The BeginLine and EndLine of an Object must use the same keyword.
```

When interpreting a Document, software MUST ignore any KeywordLine that
starts with a keyword it doesn't recognize; future implementations MUST NOT
require current clients to understand any KeywordLine not currently
described.

Other implementations that want to extend Tor's directory format MAY
introduce their own items. The keywords for extension items SHOULD start
with the characters "x-" or "X-", to guarantee that they will not conflict
with keywords used by future versions of Tor.

In our document descriptions below, we tag Items with a multiplicity in
brackets. Possible tags are:

"At start, exactly once": These items MUST occur in every instance of
the document type, and MUST appear exactly once, and MUST be the
first item in their documents.

"Exactly once": These items MUST occur exactly one time in every
instance of the document type.

"At end, exactly once": These items MUST occur in every instance of
the document type, and MUST appear exactly once, and MUST be the
last item in their documents.

"At most once": These items MAY occur zero or one times in any
instance of the document type, but MUST NOT occur more than once.

"Any number": These items MAY occur zero, one, or more times in any
instance of the document type.

"Once or more": These items MUST occur at least once in any instance
of the document type, and MAY occur more.

1.3. Signing documents

Every signable document below is signed in a similar manner, using a
given "Initial Item", a final "Signature Item", a digest algorithm, and
a signing key.

The Initial Item must be the first item in the document.

The Signature Item has the following format:

```
  <signature item keyword> [arguments] NL SIGNATURE NL
```

The "SIGNATURE" Object contains a signature (using the signing key) of
the PKCS1-padded digest of the entire document, taken from the
beginning of the Initial item, through the newline after the Signature
Item's keyword and its arguments.

Unless otherwise, the digest algorithm is SHA-1.

All documents are invalid unless signed with the correct signing key.

The "Digest" of a document, unless stated otherwise, is its digest *as
signed by this signature scheme*.

1.4. Voting timeline

Every consensus document has a "valid-after" (VA) time, a "fresh-until" (FU) time and a "valid-until" (VU) time. VA MUST precede FU, which MUST in turn precede VU. Times are chosen so that every consensus will be "fresh" until the next consensus becomes valid, and "valid" for a while after. At least 3 consensuses should be valid at any given time.

The timeline for a given consensus is as follows:

VA-DistSeconds-VoteSeconds: The authorities exchange votes.

VA-DistSeconds-VoteSeconds/2: The authorities try to download any votes they don't have.

VA-DistSeconds: The authorities calculate the consensus and exchange signatures.

VA-DistSeconds/2: The authorities try to download any signatures they don't have.

VA: All authorities have a multiply signed consensus.

VA ... FU: Caches download the consensus. (Note that since caches have no way of telling what VA and FU are until they have downloaded the consensus, they assume that the present consensus's VA is equal to the previous one's FU, and that its FU is one interval after that.)

FU: The consensus is no longer the freshest consensus.

FU ... (the current consensus's VU): Clients download the consensus. (See note above: clients guess that the next consensus's FU will be two intervals after the current VA.)

VU: The consensus is no longer valid.

VoteSeconds and DistSeconds MUST each be at least 20 seconds; FU-VA and VU-FU MUST each be at least 5 minutes.

2. Router operation and formats

2.1. Uploading server descriptors and extra-info documents

ORs SHOULD generate a new server descriptor and a new extra-info document whenever any of the following events have occurred:

- A period of time (18 hrs by default) has passed since the last time a descriptor was generated.

- A descriptor field other than bandwidth or uptime has changed.

- Bandwidth has changed by a factor of 2 from the last time a descriptor was generated, and at least a given interval of time (20 mins by default) has passed since then.

- Its uptime has been reset (by restarting).

[XXX this list is incomplete; see router_differences_are_cosmetic() in routerlist.c for others]

ORs SHOULD NOT publish a new server descriptor or extra-info document if none of the above events have occurred and not much time has passed (12 hours by default).

After generating a descriptor, ORs upload them to every directory authority they know, by posting them (in order) to the URL

```
http://<hostname:port>/tor/
```

Server descriptors may not exceed 20,000 bytes in length; extra-info documents may not exceed 50,000 bytes in length. If they do, the authorities SHOULD reject them.

2.1.1. Server descriptor format

Server descriptors consist of the following items. For backward compatibility, there should be an extra NL at the end of each router descriptor.

In lines that take multiple arguments, extra arguments SHOULD be accepted and ignored. Many of the nonterminals below are defined in section 2.1.3.

```
"router" nickname address ORPort SOCKSPort DirPort NL
```

[At start, exactly once.]

Indicates the beginning of a server descriptor. "nickname" must be a valid router nickname as specified in section 2.1.3. "address" must be an IPv4
address in dotted-quad format. The last three numbers indicate the

```
      TCP ports at which this OR exposes functionality. ORPort is a port at
      which this OR accepts TLS connections for the main OR protocol;
      SOCKSPort is deprecated and should always be 0; and DirPort is the
      port at which this OR accepts directory-related HTTP connections.  If
      any port is not supported, the value 0 is given instead of a port
      number.  (At least one of DirPort and ORPort SHOULD be set;
      authorities MAY reject any descriptor with both DirPort and ORPort of
      0.)

   "identity-ed25519" NL "-----BEGIN ED25519 CERT-----" NL certificate
           "-----END ED25519 CERT-----" NL

       [At most once, in second position in document.]

       The certificate is a base64-encoded Ed25519 certificate (see
       cert-spec.txt) terminating =s removed.  When this element is
       present, it MUST appear as the first or second element in
       the router descriptor.

       The certificate has CERT_TYPE of [04].  It must include a
       signed-with-ed25519-key extension (see cert-spec.txt,
       section 2.2.1), so that we can extract the master identity key.

    "master-key-ed25519" SP MasterKey NL

       [At most once]

       Contains the base-64 encoded ed25519 master key as a single
       argument.  If it is present, it MUST match the identity key
       in the identity-ed25519 entry.

   "bandwidth" bandwidth-avg bandwidth-burst bandwidth-observed NL

      [Exactly once]

      Estimated bandwidth for this router, in bytes per second.  The
      "average" bandwidth is the volume per second that the OR is willing to
      sustain over long periods; the "burst" bandwidth is the volume that
      the OR is willing to sustain in very short intervals.  The "observed"
      value is an estimate of the capacity this relay can handle.  The
      relay remembers the max bandwidth sustained output over any ten
      second period in the past day, and another sustained input.  The
      "observed" value is the lesser of these two numbers.

   "platform" string NL

      [At most once]

      A human-readable string describing the system on which this OR is
      running.  This MAY include the operating system, and SHOULD include
      the name and version of the software implementing the Tor protocol.

   "published" YYYY-MM-DD HH:MM:SS NL

      [Exactly once]

      The time, in UTC, when this descriptor (and its corresponding
      extra-info document if any)  was generated.

   "fingerprint" fingerprint NL

      [At most once]

      A fingerprint (a HASH_LEN-byte of asn1 encoded public key, encoded in
      hex, with a single space after every 4 characters) for this router's
      identity key. A descriptor is considered invalid (and MUST be
      rejected) if the fingerprint line does not match the public key.

      [We didn't start parsing this line until Tor 0.1.0.6-rc; it should
       be marked with "opt" until earlier versions of Tor are obsolete.]

   "hibernating" bool NL

      [At most once]

      If the value is 1, then the Tor relay was hibernating when the
      descriptor was published, and shouldn't be used to build circuits.

      [We didn't start parsing this line until Tor 0.1.0.6-rc; it should be
       marked with "opt" until earlier versions of Tor are obsolete.]

   "uptime" number NL

      [At most once]

      The number of seconds that this OR process has been running.

   "onion-key" NL a public key in PEM format

      [Exactly once]

      This key is used to encrypt CREATE cells for this OR.  The key MUST be
      accepted for at least 1 week after any new key is published in a
```

subsequent descriptor. It MUST be 1024 bits.

The key encoding is the encoding of the key as a PKCS#1 RSAPublicKey
structure, encoded in base64, and wrapped in "-----BEGIN RSA PUBLIC
KEY-----" and "-----END RSA PUBLIC KEY-----".

"onion-key-crosscert" NL a RSA signature in PEM format.

[At most once, required when identity-25519 is present]

This element contains an RSA signature, generated using the
onion-key, of the following:

```
   A SHA1 hash of the identity key  [20 bytes]
   The Ed25519 identity key [32 bytes]
```

If there is no ed25519 identity key, or if in some future version
there is no RSA identity key, the corresponding field must be
zero-filled.

Parties verifying this signature MUST allow additional data
beyond the 52 bytes listed above.

This signature proves that the party creating the descriptor
had control over the secret key corresponding to the
onion-key.

"ntor-onion-key" base-64-encoded-key

[At most once]

A curve25519 public key used for the ntor circuit extended
handshake. It's the standard encoding of the OR's curve25519
public key, encoded in base 64. The trailing = sign may be
omitted from the base64 encoding. The key MUST be accepted
for at least 1 week after any new key is published in a
subsequent descriptor.

```
"ntor-onion-key-crosscert" SP Bit NL
       "-----BEGIN ED25519 CERT-----" NL certificate
       "-----END ED25519 CERT-----" NL
```

[At most once, required when identity-25519 is present]

A signature created with the ntor-onion-key, using the
certificate format documented in cert-spec.txt, with type
[0a]. The signed key here is the master identity key.

Bit must be "0" or "1". It indicates the sign of the ed25519
public key corresponding to the ntor onion key.

To compute the ed25519 public key corresponding to a
curve25519 key, see appendix C.

This signature proves that the party creating the descriptor
had control over the secret key corresponding to the
ntor-onion-key.

"signing-key" NL a public key in PEM format

[Exactly once]

The OR's long-term RSA identity key. It MUST be 1024 bits.

The encoding is as for "onion-key" above.

"accept" exitpattern NL
"reject" exitpattern NL

[Any number]

These lines describe an "exit policy": the rules that an OR follows
when deciding whether to allow a new stream to a given address. The
'exitpattern' syntax is described below. There MUST be at least one
such entry. The rules are considered in order; if no rule matches,
the address will be accepted. For clarity, the last such entry SHOULD
be accept *:* or reject *:*.

"ipv6-policy" SP ("accept" / "reject") SP PortList NL

[At most once.]

An exit-policy summary as specified in sections 3.4.1 and 3.8.2,
summarizing
the router's rules for connecting to IPv6 addresses. A missing
"ipv6-policy" line is equivalent to "ipv6-policy reject
1-65535".

"router-sig-ed25519" SP Signature NL

[At most once]

When an identity-ed25519 element is present, there must also

```
    be a "router-sig-ed25519" element.  It MUST be the
    next-to-last element in the descriptor, appearing immediately
    before the RSA signature. It MUST contain an ed25519
    signature of a SHA256 digest of the entire document, from the
    first character up to and including the first space after the
    "router-sig-ed25519" string, prefixed with the string "Tor
    router descriptor signature v1".  Its format is:

    The signature is encoded in Base64 with terminating =s remove.

    The signing key in the identity-ed25519 certificate MUST
    be the one used to sign the document.

 "router-signature" NL Signature NL

    [At end, exactly once]

    The "SIGNATURE" object contains a signature of the PKCS1-padded
    hash of the entire server descriptor, taken from the beginning of the
    "router" line, through the newline after the "router-signature" line.
    The server descriptor is invalid unless the signature is performed
    with the router's identity key.

 "contact" info NL

    [At most once]

    Describes a way to contact the relay's administrator, preferably
    including an email address and a PGP key fingerprint.

 "family" names NL

     [At most once]

     'Names' is a space-separated list of relay nicknames or
     hexdigests. If two ORs list one another in their "family" entries,
     then OPs should treat them as a single OR for the purpose of path
     selection.

     For example, if node A's descriptor contains "family B", and node B's
     descriptor contains "family A", then node A and node B should never
     be used on the same circuit.

 "read-history" YYYY-MM-DD HH:MM:SS (NSEC s) NUM,NUM,NUM,NUM,NUM... NL
     [At most once]
 "write-history" YYYY-MM-DD HH:MM:SS (NSEC s) NUM,NUM,NUM,NUM,NUM... NL
     [At most once]

     Declare how much bandwidth the OR has used recently. Usage is divided
     into intervals of NSEC seconds.  The YYYY-MM-DD HH:MM:SS field
     defines the end of the most recent interval.  The numbers are the
     number of bytes used in the most recent intervals, ordered from
     oldest to newest.

     [We didn't start parsing these lines until Tor 0.1.0.6-rc; they should
      be marked with "opt" until earlier versions of Tor are obsolete.]

     [See also migration notes in section 2.1.2.1.]

 "eventdns" bool NL

     [At most once]

     Declare whether this version of Tor is using the newer enhanced
     dns logic.  Versions of Tor with this field set to false SHOULD NOT
     be used for reverse hostname lookups.

     [This option is obsolete.  All Tor current relays should be presumed
      to have the evdns backend.]

"caches-extra-info" NL

    [At most once.]

    Present only if this router is a directory cache that provides
    extra-info documents.

    [Versions before 0.2.0.1-alpha don't recognize this]

"extra-info-digest" digest NL

    [At most once]

    "Digest" is a hex-encoded digest (using upper-case characters) of the
    router's extra-info document, as signed in the router's extra-info
    (that is, not including the signature).  (If this field is absent, the
    router is not uploading a corresponding extra-info document.)

    [Versions before 0.2.0.1-alpha don't recognize this]

"hidden-service-dir" *(SP VersionNum) NL

    [At most once.]
```

```
        Present only if this router stores and serves hidden service
        descriptors. If any VersionNum(s) are specified, this router
        supports those descriptor versions. If none are specified, it
        defaults to version 2 descriptors.

    "protocols" SP "Link" SP LINK-VERSION-LIST SP "Circuit" SP
           CIRCUIT-VERSION-LIST NL

        [At most once.]

        Both lists are space-separated sequences of numbers, to indicate which
        protocols the server supports.  As of 30 Mar 2008, specified
        protocols are "Link 1 2 Circuit 1".  See section 4.1 of tor-spec.txt
        for more information about link protocol versions.

        [NOTE: No version of Tor uses this protocol list.  It will be removed
          in a future version of Tor.]

    "allow-single-hop-exits" NL

        [At most once.]

        Present only if the router allows single-hop circuits to make exit
        connections.  Most Tor relays do not support this: this is
        included for specialized controllers designed to support perspective
        access and such.

    "or-address" SP ADDRESS ":" PORT NL

        [Any number]

        ADDRESS = IP6ADDR | IP4ADDR
        IPV6ADDR = an ipv6 address, surrounded by square brackets.
        IPV4ADDR = an ipv4 address, represented as a dotted quad.
        PORT = a number between 1 and 65535 inclusive.

        An alternative for the address and ORPort of the "router" line, but with
        two added capabilities:

          * or-address can be either an IPv4 or IPv6 address
          * or-address allows for multiple ORPorts and addresses

        A descriptor SHOULD NOT include an or-address line that does nothing but
        duplicate the address:port pair from its "router" line.

        The ordering of or-address lines and their PORT entries matter because
        Tor MAY accept a limited number of addresses or ports. As of Tor 0.2.3.x
        only the first address and the first port are used.
```

2.1.2. Extra-info document format

```
    Extra-info documents consist of the following items:

     "extra-info" Nickname Fingerprint NL
         [At start, exactly once.]

         Identifies what router this is an extra info descriptor for.
         Fingerprint is encoded in hex (using upper-case letters), with
         no spaces.

     "identity-ed25519"
         [As in router descriptors]

     "published" YYYY-MM-DD HH:MM:SS NL

        [Exactly once.]

        The time, in UTC, when this document (and its corresponding router
        descriptor if any) was generated.  It MUST match the published time
        in the corresponding server descriptor.

     "read-history" YYYY-MM-DD HH:MM:SS (NSEC s) NUM,NUM,NUM,NUM,NUM... NL
         [At most once.]
     "write-history" YYYY-MM-DD HH:MM:SS (NSEC s) NUM,NUM,NUM,NUM,NUM... NL
         [At most once.]

         As documented in section 2.1.1 above.  See migration notes in
         section 2.1.2.1.

     "geoip-db-digest" Digest NL
         [At most once.]

         SHA1 digest of the IPv4 GeoIP database file that is used to
         resolve IPv4 addresses to country codes.

     "geoip6-db-digest" Digest NL
         [At most once.]

         SHA1 digest of the IPv6 GeoIP database file that is used to
         resolve IPv6 addresses to country codes.

     ("geoip-start-time" YYYY-MM-DD HH:MM:SS NL)
```

("geoip-client-origins" CC=N,CC=N,... NL)

Only generated by bridge routers (see blocking.pdf), and only when they have been configured with a geoip database. Non-bridges SHOULD NOT generate these fields. Contains a list of mappings from two-letter country codes (CC) to the number of clients that have connected to that bridge from that country (approximate, and rounded up to the nearest multiple of 8 in order to hamper traffic analysis). A country is included only if it has at least one address. The time in "geoip-start-time" is the time at which we began collecting geoip statistics.

"geoip-start-time" and "geoip-client-origins" have been replaced by "bridge-stats-end" and "bridge-stats-ips" in 0.2.2.4-alpha. The reason is that the measurement interval with "geoip-stats" as determined by subtracting "geoip-start-time" from "published" could have had a variable length, whereas the measurement interval in 0.2.2.4-alpha and later is set to be exactly 24 hours long. In order to clearly distinguish the new measurement intervals from the old ones, the new keywords have been introduced.

"bridge-stats-end" YYYY-MM-DD HH:MM:SS (NSEC s) NL
[At most once.]

YYYY-MM-DD HH:MM:SS defines the end of the included measurement interval of length NSEC seconds (86400 seconds by default).

A "bridge-stats-end" line, as well as any other "bridge-*" line, is only added when the relay has been running as a bridge for at least 24 hours.

"bridge-ips" CC=N,CC=N,... NL
[At most once.]

List of mappings from two-letter country codes to the number of unique IP addresses that have connected from that country to the bridge and which are no known relays, rounded up to the nearest multiple of 8.

"bridge-ip-versions" FAM=N,FAM=N,... NL
[At most once.]

List of unique IP addresses that have connected to the bridge per protocol family.

"bridge-ip-transports" PT=N,PT=N,... NL
[At most once.]

List of mappings from pluggable transport names to the number of unique IP addresses that have connected using that pluggable transport. Unobfuscated connections are counted using the reserved pluggable transport name "<OR>" (without quotes). If we received a connection from a transport proxy but we couldn't figure out the name of the pluggable transport, we use the reserved pluggable transport name "<??>".

("<OR>" and "<??>" are reserved because normal pluggable transport names MUST match the following regular expression: "[a-zA-Z_][a-zA-Z0-9_]*")

The pluggable transport name list is sorted into lexically ascending order.

If no clients have connected to the bridge yet, we only write "bridge-ip-transports" to the stats file.

"dirreq-stats-end" YYYY-MM-DD HH:MM:SS (NSEC s) NL
[At most once.]

YYYY-MM-DD HH:MM:SS defines the end of the included measurement interval of length NSEC seconds (86400 seconds by default).

A "dirreq-stats-end" line, as well as any other "dirreq-*" line, is only added when the relay has opened its Dir port and after 24 hours of measuring directory requests.

"dirreq-v2-ips" CC=N,CC=N,... NL
[At most once.]
"dirreq-v3-ips" CC=N,CC=N,... NL
[At most once.]

List of mappings from two-letter country codes to the number of unique IP addresses that have connected from that country to request a v2/v3 network status, rounded up to the nearest multiple of 8. Only those IP addresses are counted that the directory can answer with a 200 OK status code. (Note here and below: current Tor versions, as of 0.2.5.2-alpha, no longer cache or serve v2 networkstatus documents.)

"dirreq-v2-reqs" CC=N,CC=N,... NL
[At most once.]

```
"dirreq-v3-reqs" CC=N,CC=N,... NL
    [At most once.]

    List of mappings from two-letter country codes to the number of
    requests for v2/v3 network statuses from that country, rounded up
    to the nearest multiple of 8. Only those requests are counted that
    the directory can answer with a 200 OK status code.

"dirreq-v2-share" num% NL
    [At most once.]
"dirreq-v3-share" num% NL
    [At most once.]

    The share of v2/v3 network status requests that the directory
    expects to receive from clients based on its advertised bandwidth
    compared to the overall network bandwidth capacity. Shares are
    formatted in percent with two decimal places. Shares are
    calculated as means over the whole 24-hour interval.

"dirreq-v2-resp" status=num,... NL
    [At most once.]
"dirreq-v3-resp" status=nul,... NL
    [At most once.]

    List of mappings from response statuses to the number of requests
    for v2/v3 network statuses that were answered with that response
    status, rounded up to the nearest multiple of 4. Only response
    statuses with at least 1 response are reported. New response
    statuses can be added at any time. The current list of response
    statuses is as follows:

    "ok": a network status request is answered; this number
       corresponds to the sum of all requests as reported in
       "dirreq-v2-reqs" or "dirreq-v3-reqs", respectively, before
       rounding up.
    "not-enough-sigs: a version 3 network status is not signed by a
       sufficient number of requested authorities.
    "unavailable": a requested network status object is unavailable.
    "not-found": a requested network status is not found.
    "not-modified": a network status has not been modified since the
       If-Modified-Since time that is included in the request.
    "busy": the directory is busy.

"dirreq-v2-direct-dl" key=val,... NL
    [At most once.]
"dirreq-v3-direct-dl" key=val,... NL
    [At most once.]
"dirreq-v2-tunneled-dl" key=val,... NL
    [At most once.]
"dirreq-v3-tunneled-dl" key=val,... NL
    [At most once.]

    List of statistics about possible failures in the download process
    of v2/v3 network statuses. Requests are either "direct"
    HTTP-encoded requests over the relay's directory port, or
    "tunneled" requests using a BEGIN_DIR cell over the relay's OR
    port. The list of possible statistics can change, and statistics
    can be left out from reporting. The current list of statistics is
    as follows:

    Successful downloads and failures:

    "complete": a client has finished the download successfully.
    "timeout": a download did not finish within 10 minutes after
       starting to send the response.
    "running": a download is still running at the end of the
       measurement period for less than 10 minutes after starting to
       send the response.

    Download times:

    "min", "max": smallest and largest measured bandwidth in B/s.
    "d[1-4,6-9]": 1st to 4th and 6th to 9th decile of measured
       bandwidth in B/s. For a given decile i, i/10 of all downloads
       had a smaller bandwidth than di, and (10-i)/10 of all downloads
       had a larger bandwidth than di.
    "q[1,3]": 1st and 3rd quartile of measured bandwidth in B/s. One
       fourth of all downloads had a smaller bandwidth than q1, one
       fourth of all downloads had a larger bandwidth than q3, and the
       remaining half of all downloads had a bandwidth between q1 and
       q3.
    "md": median of measured bandwidth in B/s. Half of the downloads
       had a smaller bandwidth than md, the other half had a larger
       bandwidth than md.

"dirreq-read-history" YYYY-MM-DD HH:MM:SS (NSEC s) NUM,NUM,NUM... NL
    [At most once]
"dirreq-write-history" YYYY-MM-DD HH:MM:SS (NSEC s) NUM,NUM,NUM... NL
    [At most once]

    Declare how much bandwidth the OR has spent on answering directory
    requests.  Usage is divided into intervals of NSEC seconds.  The
    YYYY-MM-DD HH:MM:SS field defines the end of the most recent
```

interval. The numbers are the number of bytes used in the most recent intervals, ordered from oldest to newest.

"entry-stats-end" YYYY-MM-DD HH:MM:SS (NSEC s) NL
[At most once.]

YYYY-MM-DD HH:MM:SS defines the end of the included measurement interval of length NSEC seconds (86400 seconds by default).

An "entry-stats-end" line, as well as any other "entry-*" line, is first added after the relay has been running for at least 24 hours.

"entry-ips" CC=N,CC=N,... NL
[At most once.]

List of mappings from two-letter country codes to the number of unique IP addresses that have connected from that country to the relay and which are no known other relays, rounded up to the nearest multiple of 8.

"cell-stats-end" YYYY-MM-DD HH:MM:SS (NSEC s) NL
[At most once.]

YYYY-MM-DD HH:MM:SS defines the end of the included measurement interval of length NSEC seconds (86400 seconds by default).

A "cell-stats-end" line, as well as any other "cell-*" line, is first added after the relay has been running for at least 24 hours.

"cell-processed-cells" num,...,num NL
[At most once.]

Mean number of processed cells per circuit, subdivided into deciles of circuits by the number of cells they have processed in descending order from loudest to quietest circuits.

"cell-queued-cells" num,...,num NL
[At most once.]

Mean number of cells contained in queues by circuit decile. These means are calculated by 1) determining the mean number of cells in a single circuit between its creation and its termination and 2) calculating the mean for all circuits in a given decile as determined in "cell-processed-cells". Numbers have a precision of two decimal places.

Note that this statistic can be inaccurate for circuits that had queued cells at the start or end of the measurement interval.

"cell-time-in-queue" num,...,num NL
[At most once.]

Mean time cells spend in circuit queues in milliseconds. Times are calculated by 1) determining the mean time cells spend in the queue of a single circuit and 2) calculating the mean for all circuits in a given decile as determined in "cell-processed-cells".

Note that this statistic can be inaccurate for circuits that had queued cells at the start or end of the measurement interval.

"cell-circuits-per-decile" num NL
[At most once.]

Mean number of circuits that are included in any of the deciles, rounded up to the next integer.

"conn-bi-direct" YYYY-MM-DD HH:MM:SS (NSEC s) BELOW,READ,WRITE,BOTH NL
[At most once]

Number of connections, split into 10-second intervals, that are used uni-directionally or bi-directionally as observed in the NSEC seconds (usually 86400 seconds) before YYYY-MM-DD HH:MM:SS. Every 10 seconds, we determine for every connection whether we read and wrote less than a threshold of 20 KiB (BELOW), read at least 10 times more than we wrote (READ), wrote at least 10 times more than we read (WRITE), or read and wrote more than the threshold, but not 10 times more in either direction (BOTH). After classifying a connection, read and write counters are reset for the next 10-second interval.

"exit-stats-end" YYYY-MM-DD HH:MM:SS (NSEC s) NL
[At most once.]

YYYY-MM-DD HH:MM:SS defines the end of the included measurement interval of length NSEC seconds (86400 seconds by default).

An "exit-stats-end" line, as well as any other "exit-*" line, is first added after the relay has been running for at least 24 hours and only if the relay permits exiting (where exiting to a single port and IP address is sufficient).

```
    "exit-kibibytes-written" port=N,port=N,... NL
        [At most once.]
    "exit-kibibytes-read" port=N,port=N,... NL
        [At most once.]

        List of mappings from ports to the number of kibibytes that the
        relay has written to or read from exit connections to that port,
        rounded up to the next full kibibyte.  Relays may limit the
        number of listed ports and subsume any remaining kibibytes under
        port "other".

    "exit-streams-opened" port=N,port=N,... NL
        [At most once.]

        List of mappings from ports to the number of opened exit streams
        to that port, rounded up to the nearest multiple of 4.  Relays may
        limit the number of listed ports and subsume any remaining opened
        streams under port "other".

    "hidserv-stats-end" YYYY-MM-DD HH:MM:SS (NSEC s) NL
        [At most once.]

        YYYY-MM-DD HH:MM:SS defines the end of the included measurement
        interval of length NSEC seconds (86400 seconds by default).

        A "hidserv-stats-end" line, as well as any other "hidserv-*" line,
        is first added after the relay has been running for at least 24
        hours.

    "hidserv-rend-relayed-cells" SP num SP key=val SP key=val ... NL
        [At most once.]

        Approximate number of RELAY cells seen in either direction on a
        circuit after receiving and successfully processing a RENDEZVOUS1
        cell.

        The original measurement value is obfuscated in several steps:
        first, it is rounded up to the nearest multiple of 'bin_size'
        which is reported in the key=val part of this line; second, a
        (possibly negative) noise value is added to the result of the
        first step by randomly sampling from a Laplace distribution with
        mu = 0 and b = (delta_f / epsilon) with 'delta_f' and 'epsilon'
        being reported in the key=val part, too; third, the result of the
        previous obfuscation steps is truncated to the next smaller
        integer and included as 'num'. Note that the overall reported
        value can be negative.

    "hidserv-dir-onions-seen" SP num SP key=val SP key=val ... NL
        [At most once.]

        Approximate number of unique hidden-service identities seen in
        descriptors published to and accepted by this hidden-service
        directory.

        The original measurement value is obfuscated in the same way as
        the 'num' value reported in "hidserv-rend-relayed-cells", but
        possibly with different parameters as reported in the key=val part
        of this line. Note that the overall reported value can be
        negative.

    "transport" transportname address:port [arglist] NL
        [Any number.]

        Signals that the router supports the 'transportname' pluggable
        transport in IP address 'address' and TCP port 'port'. A single
        descriptor MUST not have more than one transport line with the
        same 'transportname'.

        Pluggable transports are only relevant to bridges, but these entries
        can appear in non-bridge relays as well.

    "router-sig-ed25519"
        [As in router descriptors]

    "router-signature" NL Signature NL
        [At end, exactly once.]

        A document signature as documented in section 1.3, using the
        initial item "extra-info" and the final item "router-signature",
        signed with the router's identity key.
```

2.1.2.1. Moving history fields to extra-info documents

Tools that want to use the read-history and write-history values SHOULD download extra-info documents as well as server descriptors. Such tools SHOULD accept history values from both sources; if they appear in both documents, the values in the extra-info documents are authoritative.

New versions of Tor no longer generate server descriptors containing read-history or write-history. Tools should continue to accept read-history and write-history values in server descriptors produced by older versions of Tor until all Tor versions earlier than 0.2.0.x are obsolete.

2.1.3. Nonterminals in server descriptors

```
   nickname ::= between 1 and 19 alphanumeric characters ([A-Za-z0-9]),
      case-insensitive.
   hexdigest ::= a '$', followed by 40 hexadecimal characters
      ([A-Fa-f0-9]). [Represents a relay by the digest of its identity
      key.]

   exitpattern ::= addrspec ":" portspec
   portspec ::= "*" | port | port "-" port
   port ::= an integer between 1 and 65535, inclusive.

      [Some implementations incorrectly generate ports with value 0.
       Implementations SHOULD accept this, and SHOULD NOT generate it.
       Connections to port 0 are never permitted.]

   addrspec ::= "*" | ip4spec | ip6spec
   ipv4spec ::= ip4 | ip4 "/" num_ip4_bits | ip4 "/" ip4mask
   ip4 ::= an IPv4 address in dotted-quad format
   ip4mask ::= an IPv4 mask in dotted-quad format
   num_ip4_bits ::= an integer between 0 and 32
   ip6spec ::= ip6 | ip6 "/" num_ip6_bits
   ip6 ::= an IPv6 address, surrounded by square brackets.
   num_ip6_bits ::= an integer between 0 and 128

   bool ::= "0" | "1"
```

3. Directory authority operation and formats

Every authority has two keys used in this protocol: a signing key, and an authority identity key. (Authorities also have a router identity key used in their role as a router and by earlier versions of the directory protocol.) The identity key is used from time to time to sign new key certificates using new signing keys; it is very sensitive. The signing key is used to sign key certificates and status documents.

3.1. Creating key certificates

Key certificates consist of the following items:

```
    "dir-key-certificate-version" version NL

        [At start, exactly once.]

        Determines the version of the key certificate.  MUST be "3" for
        the protocol described in this document.  Implementations MUST
        reject formats they don't understand.

    "dir-address" IPPort NL
        [At most once]

        An IP:Port for this authority's directory port.

    "fingerprint" fingerprint NL

        [Exactly once.]

        Hexadecimal encoding without spaces based on the authority's
        identity key.

    "dir-identity-key" NL a public key in PEM format

        [Exactly once.]

        The long-term authority identity key for this authority.  This key
        SHOULD be at least 2048 bits long; it MUST NOT be shorter than
        1024 bits.

    "dir-key-published" YYYY-MM-DD HH:MM:SS NL

        [Exactly once.]

        The time (in UTC) when this document and corresponding key were
        last generated.

    "dir-key-expires" YYYY-MM-DD HH:MM:SS NL

        [Exactly once.]

        A time (in UTC) after which this key is no longer valid.

    "dir-signing-key" NL a key in PEM format

        [Exactly once.]

        The directory server's public signing key.  This key MUST be at
        least 1024 bits, and MAY be longer.

    "dir-key-crosscert" NL CrossSignature NL

        [Exactly once.]
        CrossSignature is a signature, made using the certificate's signing
        key, of the digest of the PKCS1-padded hash of the certificate's
        identity key.  For backward compatibility with broken versions of the
        parser, we wrap the base64-encoded signature in -----BEGIN ID
        SIGNATURE---- and -----END ID SIGNATURE----- tags.  Implementations
        MUST allow the "ID " portion to be omitted, however.

        Implementations MUST verify that the signature is a correct signature
        of the hash of the identity key using the signing key.

    "dir-key-certification" NL Signature NL

        [At end, exactly once.]

        A document signature as documented in section 1.3, using the
        initial item "dir-key-certificate-version" and the final item
        "dir-key-certification", signed with the authority identity key.

   Authorities MUST generate a new signing key and corresponding
   certificate before the key expires.
```

3.2. Accepting server descriptor and extra-info document uploads

```
   When a router posts a signed descriptor to a directory authority, the
   authority first checks whether it is well-formed and correctly
   self-signed.  If it is, the authority next verifies that the nickname
   in question is not already assigned to a router with a different
   public key.
   Finally, the authority MAY check that the router is not blacklisted
   because of its key, IP, or another reason.

   An authority also keeps a record of all the Ed25519/RSA1024
   identity key pairs that it has seen before.  It rejects any
   descriptor that has a known Ed/RSA identity key that it has
   already seen accompanied by a different RSA/Ed identity key
   in an older descriptor.

   At a future date, authorities will begin rejecting all
   descriptors whose RSA key was previously accompanied by an
   Ed25519 key, if the descriptor does not list and Ed25519 key.

   At a future date, authorities will begin rejecting all descriptors
   that do not list an Ed25519 key.

   If the descriptor passes these tests, and the authority does not already
   have a descriptor for a router with this public key, it accepts the
   descriptor and remembers it.

   If the authority _does_ have a descriptor with the same public key, the
   newly uploaded descriptor is remembered if its publication time is more
   recent than the most recent old descriptor for that router, and either:
      - There are non-cosmetic differences between the old descriptor and the
        new one.
      - Enough time has passed between the descriptors' publication times.
        (Currently, 12 hours.)

   Differences between server descriptors are "non-cosmetic" if they would be
   sufficient to force an upload as described in section 2.1 above.

   Note that the "cosmetic difference" test only applies to uploaded
   descriptors, not to descriptors that the authority downloads from other
   authorities.

   When a router posts a signed extra-info document to a directory authority,
   the authority again checks it for well-formedness and correct signature,
   and checks that its matches the extra-info-digest in some router
   descriptor that it believes is currently useful.  If so, it accepts it and
   stores it and serves it as requested.  If not, it drops it.
```

3.3. Computing microdescriptors

```
   Microdescriptors are a stripped-down version of server descriptors
   generated by the directory authorities which may additionally contain
   authority-generated information.  Microdescriptors contain only the
   most relevant parts that clients care about.  Microdescriptors are
   expected to be relatively static and only change about once per week.
   Microdescriptors do not contain any information that clients need to
   use to decide which servers to fetch information about, or which
   servers to fetch information from.

   Microdescriptors are a straight transform from the server descriptor
   and the consensus method.  Microdescriptors have no header or footer.
   Microdescriptors are identified by the hash of its concatenated
   elements without a signature by the router.  Microdescriptors do not
   contain any version information, because their version is determined
   by the consensus method.

   Starting with consensus method 8, microdescriptors contain the
   following elements taken from or based on the server descriptor.  Order
   matters here, because different directory authorities must be able to
   transform a given server descriptor and consensus method into the exact
```

same microdescriptor.

"onion-key" NL a public key in PEM format

[Exactly once, at start]

The "onion-key" element as specified in section 2.1.1.

"ntor-onion-key" SP base-64-encoded-key NL

[At most once]

The "ntor-onion-key" element as specified in section 2.1.1.

(Only included when generating microdescriptors for
consensus-method 16 or later.)

"a" SP address ":" port NL

[Any number]

The "or-address" element as specified in section 2.1.1.

"family" names NL

[At most once]

The "family" element as specified in section 2.1.1.

"p" SP ("accept" / "reject") SP PortList NL

[At most once]

The exit-policy summary as specified in sections 3.4.1 and 3.8.2. A
missing "p" line is equivalent to "p reject 1-65535".

[With microdescriptors, clients don't learn exact exit policies:
clients can only guess whether a relay accepts their request, try the
BEGIN request, and might get end-reason-exit-policy if they guessed
wrong, in which case they'll have to try elsewhere.]

"p6" SP ("accept" / "reject") SP PortList NL

[At most once]

The IPv6 exit policy summary as specified in sections 3.4.1 and 3.8.2. A
missing "p6" line is equivalent to "p6 reject 1-65535".

(Only included when generating microdescriptors for
consensus-method 15 or later.)

"id" SP "rsa1024" SP base64-encoded-identity-digest NL

[At most once]

The node identity digest (as described in tor-spec.txt), base64
encoded, without trailing =s. This line is included to prevent
collisions between microdescriptors.

Implementations SHOULD ignore these lines: they are
added to microdescriptors only to prevent collisions.

(Only included when generating microdescriptors for
consensus-method 18 or later.)

"id" SP "ed25519" SP base64-encoded-ed25519-identity NL

[At most once]

The node's master Ed25519 identity key, base64 encoded,
without trailing =s.

(Only included when generating microdescriptors for
consensus-method 21 or later.)

"id" SP keytype ... NL

[At most once per distinct keytype.]

Implementations MUST ignore "id" lines with unrecognized
key-types in place of "rsa1024" or "ed25519"

(Note that with microdescriptors, clients do not learn the RSA identity of
their routers: they only learn a hash of the RSA identity key. This is
all they need to confirm the actual identity key when doing a TLS
handshake, and all they need to put the identity key digest in their
CREATE cells.)

3.4. Exchanging votes

Authorities divide time into Intervals. Authority administrators SHOULD
try to all pick the same interval length, and SHOULD pick intervals that

are commonly used divisions of time (e.g., 5 minutes, 15 minutes, 30 minutes, 60 minutes, 90 minutes). Voting intervals SHOULD be chosen to divide evenly into a 24-hour day.

Authorities SHOULD act according to interval and delays in the latest consensus. Lacking a latest consensus, they SHOULD default to a 30-minute Interval, a 5 minute VotingDelay, and a 5 minute DistDelay.

Authorities MUST take pains to ensure that their clocks remain accurate within a few seconds. (Running NTP is usually sufficient.)

The first voting period of each day begins at 00:00 (midnight) UTC. If the last period of the day would be truncated by one-half or more, it is merged with the second-to-last period.

An authority SHOULD publish its vote immediately at the start of each voting period (minus VoteSeconds+DistSeconds). It does this by making it available at

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

and sending it in an HTTP POST request to each other authority at the URL

```
  http://<hostname>/tor/post/vote
```

If, at the start of the voting period, minus DistSeconds, an authority does not have a current statement from another authority, the first authority downloads the other's statement.

Once an authority has a vote from another authority, it makes it available at

```
   http://<hostname>/tor/status-vote/next/<fp>.z
```

where <fp> is the fingerprint of the other authority's identity key. And at

```
   http://<hostname>/tor/status-vote/next/d/<d>.z
```

where <d> is the digest of the vote document.

Also, once an authority receives a vote from another authority, it examines it for new descriptors and fetches them from that authority. This may the only way for an authority to hear about relays that didn't publish their descriptor to all authorities, and while it's too late for the authority to include relays in its current vote, it can include them in its next vote. See section 3.6 below for details.

3.4.1. Vote and consensus status document formats

Votes and consensuses are more strictly formatted than other documents in this specification, since different authorities must be able to generate exactly the same consensus given the same set of votes.

The procedure for deciding when to generate vote and consensus status documents are described in section 1.4 on the voting timeline.

Status documents contain a preamble, an authority section, a list of router status entries, and one or more footer signature, in that order.

Unlike other formats described above, a SP in these documents must be a single space character (hex 20).

Some items appear only in votes, and some items appear only in consensuses. Unless specified, items occur in both.

The preamble contains the following items. They MUST occur in the order given here:

```
 "network-status-version" SP version NL

     [At start, exactly once.]

     A document format version.  For this specification, the version is
     "3".

 "vote-status" SP type NL

     [Exactly once.]

     The status MUST be "vote" or "consensus", depending on the type of
     the document.

 "consensus-methods" SP IntegerList NL

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

     A space-separated list of supported methods for generating
     consensuses from votes.  See section 3.8.1 for details.  Absence of
     the line means that only method "1" is supported.

 "consensus-method" SP Integer NL

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

     See section 3.8.1 for details.

     (Only included when the vote is generated with consensus-method 2 or
     later.)
```

```
"published" SP YYYY-MM-DD SP HH:MM:SS NL

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

    The publication time for this status document (if a vote).

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

    [Exactly once.]

    The start of the Interval for this vote.  Before this time, the
    consensus document produced from this vote should not be used.
    See section 1.4 for voting timeline information.

"flag-thresholds" SP Thresholds NL

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

     A space-separated list of the internal performance thresholds
     that the directory authority had at the moment it was forming
     a vote.

     The metaformat is:
        Thresholds = Threshold | Threshold SP Thresholds
        Threshold = ThresholdKey '=' ThresholdVal
        ThresholdKey = (KeywordChar | "_") +
        ThresholdVal = [0-9]+("."[0-9]+)? "%"?

     Commonly used Thresholds at this point include:

     "stable-uptime" -- Uptime (in seconds) required for a relay
                        to be marked as stable.

     "stable-mtbf" -- MTBF (in seconds) required for a relay to be
                      marked as stable.

     "enough-mtbf" -- Whether we have measured enough MTBF to look
                      at stable-mtbf instead of stable-uptime.

     "fast-speed" -- Bandwidth (in bytes per second) required for
                     a relay to be marked as fast.

     "guard-wfu" -- WFU (in seconds) required for a relay to be
                    marked as guard.

     "guard-tk" -- Weighted Time Known (in seconds) required for a
                   relay to be marked as guard.

     "guard-bw-inc-exits" -- If exits can be guards, then all guards
                             must have a bandwidth this high.

     "guard-bw-exc-exits" -- If exits can't be guards, then all guards
                             must have a bandwidth this high.

     "ignoring-advertised-bws" -- 1 if we have enough measured bandwidths
                             that we'll ignore the advertised bandwidth
                             claims of routers without measured bandwidth.

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

    [Exactly once.]

    The time at which the next consensus should be produced; before this
    time, there is no point in downloading another consensus, since there
    won't be a new one.  See section 1.4 for voting timeline information.

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

    [Exactly once.]

    The end of the Interval for this vote.  After this time, the
    consensus produced by this vote should not be used.  See section 1.4
    for voting timeline information.

"voting-delay" SP VoteSeconds SP DistSeconds NL

    [Exactly once.]

    VoteSeconds is the number of seconds that we will allow to collect
    votes from all authorities; DistSeconds is the number of seconds
    we'll allow to collect signatures from all authorities. See
    section 1.4 for voting timeline information.

"client-versions" SP VersionList NL

    [At most once.]

    A comma-separated list of recommended Tor versions for client
    usage, in ascending order. The versions are given as defined by
    version-spec.txt. If absent, no opinion is held about client
    versions.

"server-versions" SP VersionList NL
```

```
        [At most once.]

        A comma-separated list of recommended Tor versions for relay
        usage, in ascending order. The versions are given as defined by
        version-spec.txt. If absent, no opinion is held about server
        versions.

    "package" SP PackageName SP Version SP URL SP DIGESTS NL

        [Any number of times.]

        For this element:

        PACKAGENAME = NONSPACE
        VERSION = NONSPACE
        URL = NONSPACE
        DIGESTS = DIGEST | DIGESTS SP DIGEST
        DIGEST = DIGESTTYPE "=" DIGESTVAL
        NONSPACE = one or more non-space printing characters
        DIGESTVAL = DIGESTTYPE = one or more non-=, non-" " characters.

        Indicates that a package called "package" of version VERSION may be
        found at URL, and its digest as computed with DIGESTTYPE is equal to
        DIGESTVAL.  In consensuses, these lines are sorted lexically by
        "PACKAGENAME VERSION" pairs, and DIGESTTYPES must appear in ascending
        order.  A consensus must not contain the same "PACKAGENAME VERSION"
        more than once.  If a vote contains the same "PACKAGENAME VERSION"
        more than once, all but the last is ignored.

        Included in consensuses only for method 19 and later.

    "known-flags" SP FlagList NL

        [Exactly once.]

        A space-separated list of all of the flags that this document
        might contain.  A flag is "known" either because the authority
        knows about them and might set them (if in a vote), or because
        enough votes were counted for the consensus for an authoritative
        opinion to have been formed about their status.

    "params" SP [Parameters] NL

        [At most once]

        Parameter ::= Keyword '=' Int32
        Int32 ::= A decimal integer between -2147483648 and 2147483647.
        Parameters ::= Parameter | Parameters SP Parameter

        The parameters list, if present, contains a space-separated list of
        case-sensitive key-value pairs, sorted in lexical order by their
        keyword (as ASCII byte strings). Each parameter has its own meaning.

        (Only included when the vote is generated with consensus-method 7 or
        later.)

        Commonly used "param" arguments at this point include:

        "circwindow" -- the default package window that circuits should
        be established with. It started out at 1000 cells, but some
        research indicates that a lower value would mean fewer cells in
        transit in the network at any given time.
        Min: 100, Max: 1000
        First-appeared: Tor 0.2.1.20

        "CircuitPriorityHalflifeMsec" -- the halflife parameter used when
        weighting which circuit will send the next cell. Obeyed by Tor
        0.2.2.10-alpha and later.  (Versions of Tor between 0.2.2.7-alpha
        and 0.2.2.10-alpha recognized a "CircPriorityHalflifeMsec" parameter,
        but mishandled it badly.)
        Min: -1, Max: 2147483647 (INT32_MAX)
        First-appeared: Tor 0.2.2.11-alpha

        "perconnbwrate" and "perconnbwburst" -- if set, each relay sets
        up a separate token bucket for every client OR connection,
        and rate limits that connection indepedently. Typically left
        unset, except when used for performance experiments around trac
        entry 1750. Only honored by relays running Tor 0.2.2.16-alpha
        and later. (Note that relays running 0.2.2.7-alpha through
        0.2.2.14-alpha looked for bwconnrate and bwconnburst, but then
        did the wrong thing with them; see bug 1830 for details.)
        Min: 1, Max: 2147483647 (INT32_MAX)
        First-appeared: 0.2.2.7-alpha
        Removed-in: 0.2.2.16-alpha

        "refuseunknownexits" -- if set to one, exit relays look at
        the previous hop of circuits that ask to open an exit stream,
        and refuse to exit if they don't recognize it as a relay. The
        goal is to make it harder for people to use them as one-hop
        proxies. See trac entry 1751 for details.
        Min: 0, Max: 1
        First-appeared: 0.2.2.17-alpha
```

```
     "bwweightscale" -- Value that bandwidth-weights are divided by. If not
     present then this defaults to 10000.
     Min: 1
     First-appeared: 0.2.2.10-alpha

     "cbtdisabled", "cbtnummodes", "cbtrecentcount", "cbtmaxtimeouts",
     "cbtmincircs", "cbtquantile", "cbtclosequantile", "cbttestfreq",
     "cbtmintimeout", and "cbtinitialtimeout" -- see "2.4.5. Consensus
     parameters governing behavior" in path-spec.txt for a series of
     circuit build time related consensus params.

     "UseOptimisticData" -- If set to zero, clients by default
     shouldn't try to send optimistic data to servers until they have
     received a RELAY_CONNECTED cell.
     Min: 0, Max: 1, Default: 0
     First-appeared: 0.2.3.3-alpha

     "maxunmeasuredbw" -- Used by authorities during voting with
     method 17 or later. The maximum value to give for any Bandwidth=
     entry for a router that isn't based on at least three
     measurements.
     First-appeared: 0.2.4.11-alpha

     "Support022HiddenServices" -- Used to implement a mass switch-over
     from sending timestamps to hidden services by default to sending
     no timestamps at all.  If this option is absent, or is set to 1,
     clients with the default configuration send timestamps; otherwise,
     they do not.
     Min: 0, Max: 1. Default: 1.
     First-appeared: 0.2.4.18-rc

     "usecreatefast" -- Used to control whether clients use the
     CREATE_FAST handshake on the first hop of their circuits.
     Min: 0, Max: 1. Default: 1.
     First-appeared: 0.2.4.23, 0.2.5.2-alpha

     "pb_mincircs", "pb_noticepct", "pb_warnpct", "pb_extremepct",
     "pb_dropguards", "pb_scalecircs", "pb_scalefactor",
     "pb_multfactor", "pb_minuse", "pb_noticeusepct",
     "pb_extremeusepct", "pb_scaleuse" -- DOCDOC

     "UseNTorHandshake" -- If true, then versions of Tor that support
       NTor will prefer to use it by default.
     Min: 0,  Max: 1. Default: 1.
     First-appeared: 0.2.4.8-alpha

     "FastFlagMinThreshold", "FastFlagMaxThreshold" -- lowest and
     highest allowable values for the cutoff for routers that should get
     the Fast flag.  This is used during voting to prevent the threshold
     for getting the Fast flag from being too low or too high.
     FastFlagMinThreshold: Min: 4. Max: INT32_MAX: Default: 4.
     FastFlagMaxThreshold: Min: -. Max: INT32_MAX: Default: INT32_MAX
     First-appeared: 0.2.3.11-alpha

     "NumDirectoryGuards", "NumEntryGuards" -- Number of guard nodes
     clients should use by default.  If NumDirectoryGuards is 0,
     we default to NumEntryGuards.
     NumDirectoryGuards: Min: 0. Max: 10. Default: 0
     NumEntryGuards:     Min: 1. Max: 10. Default: 3
     First-appeared: 0.2.4.23, 0.2.5.6-alpha

     "GuardLifetime" -- Duration for which clients should choose guard
     nodes, in seconds.
     Min: 30 days.  Max: 1826 days.  Default: 60 days.
     First-appeared: 0.2.4.12-alpha

     "min_paths_for_circs_pct" -- DOCDOC

     "NumNTorsPerTAP" -- When balancing ntor and TAP cells at relays,
     how many ntor handshakes should we perform for each TAP handshake?
     Min: 1. Max: 100000. Default: 10.
     First-appeared: 0.2.4.17-rc

     "AllowNonearlyExtend" -- If true, permit EXTEND cells that are not
     inside RELAY_EARLY cells.
     Min: 0. Max: 1. Default: 0.
     First-appeared: 0.2.3.11-alpha

The authority section of a vote contains the following items, followed
in turn by the authority's current key certificate:

 "dir-source" SP nickname SP identity SP address SP IP SP dirport SP
    orport NL

     [Exactly once, at start]

     Describes this authority.  The nickname is a convenient identifier
     for the authority.  The identity is an uppercase hex fingerprint of
     the authority's current (v3 authority) identity key.  The address is
     the server's hostname.  The IP is the server's current IP address,
     and dirport is its current directory port. XXXXorport
 "contact" SP string NL

     [Exactly once.]

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

 "legacy-dir-key" SP FINGERPRINT NL

     [At most once]

     Lists a fingerprint for an obsolete _identity_ key still used
     by this authority to keep older clients working.  This option
     is 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.)

The authority section of a consensus contains groups the following items,
in the order given, with one group for each authority that contributed to
the consensus, with groups sorted by authority identity digest:

 "dir-source" SP nickname SP identity SP address SP IP SP dirport SP
    orport NL

     [Exactly once, at start]

     As in the authority section of a vote.

 "contact" SP string NL

     [Exactly once.]

     As in the authority section of a vote.

 "vote-digest" SP digest NL

     [Exactly once.]

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

For each "legacy-dir-key" in the vote, there is an additional "dir-source"
line containing that legacy key's fingerprint, the authority's nickname
with "-legacy" appended, and all other fields as in the main "dir-source"
line for that authority.  These "dir-source" lines do not have
corresponding "contact" or "vote-digest" entries.

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

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

     [At start, exactly once.]

     "Nickname" is the OR's nickname.  "Identity" is a hash of its
     identity key, encoded in base64, with trailing equals sign(s)
     removed.  "Digest" is a hash of its most recent descriptor as
     signed (that is, not including the signature), encoded in base64.
     "Publication" is the
     publication time of its most recent descriptor, in the form
     YYYY-MM-DD HH:MM:SS, in UTC.  "IP" is its current IP address;
     ORPort is its current OR port, "DirPort" is its current directory
     port, or "0" for "none".

 "a" SP address ":" port NL

     [Any number.]

     Present only if the OR has at least one IPv6 address.

     Address and portlist are as for "or-address" as specified in
     section 2.1.1.

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

 "s" SP Flags NL

     [Exactly once.]

     A series of space-separated status flags, in lexical order (as ASCII
     byte strings).  Currently documented flags are:

       "Authority" if the router is a directory authority.
       "BadExit" if the router 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" if the router is more useful for building
```

```
          general-purpose exit circuits than for relay circuits.  The
          path building algorithm uses this flag; see path-spec.txt.
       "Fast" if the router is suitable for high-bandwidth circuits.
       "Guard" if the router is suitable for use as an entry guard.
       "HSDir" if the router is considered a v2 hidden service directory.
       "Named" if the router's identity-nickname mapping is canonical,
          and this authority binds names.
       "Stable" if the router is suitable for long-lived circuits.
       "Running" if the router is currently usable.
       "Unnamed" if another router has bound the name used by this
          router, and this authority binds names.
       "Valid" if the router has been 'validated'.
       "V2Dir" if the router implements the v2 directory protocol or
          higher.

 "v" SP version NL

     [At most once.]

     The version of the Tor protocol that this relay is running.  If
     the value 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, if the value begins with
     some other string, 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 if they would cause "v" lines to be over 128 characters
     long.

 "w" SP "Bandwidth=" INT [SP "Measured=" INT] [SP "Unmeasured=1"] NL

     [At most once.]

     An estimate of the bandwidth of this relay, in an arbitrary
     unit (currently kilobytes per second).  Used to weight router
     selection. See section 3.4.2 for details on how the value of
     Bandwidth is determined in a consensus.

     Additionally, the Measured= keyword is present in votes by
     participating bandwidth measurement authorities to indicate
     a measured bandwidth currently produced by measuring stream
     capacities. It does not occur in consensuses.

     The "Unmeasured=1" value is included in consensuses generated
     with method 17 or later when the 'Bandwidth=' value is not
     based on a threshold of 3 or more measurements for this relay.

     Other weighting keywords may be added later.
     Clients MUST ignore keywords they do not recognize.

 "p" SP ("accept" / "reject") SP PortList NL

     [At most once.]

     PortList = PortOrRange
     PortList = PortList "," PortOrRange
     PortOrRange = INT "-" INT / INT

     A list of those ports that this router supports (if 'accept')
     or does not support (if 'reject') for exit to "most
     addresses".

  "m" SP methods 1*(SP algorithm "=" digest) NL

     [Any number, only in votes.]

     Microdescriptor hashes for all consensus methods that an authority
     supports and that use the same microdescriptor format.  "methods"
     is a comma-separated list of the consensus methods that the
     authority believes will produce "digest".  "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 the base64 encoding of the hash of
     the router's microdescriptor with trailing =s omitted.

  "id" SP "ed25519" SP ed25519-identity NL
  "id" SP "ed25519" SP "none" NL
     [vote only, at most once]

The footer section is delineated in all votes and consensuses supporting
consensus method 9 and above with the following:

 "directory-footer" NL

It contains two subsections, a bandwidths-weights line and a
directory-signature. (Prior to conensus method 9, footers only contained
directory-signatures without a 'directory-footer' line or
bandwidth-weights.)

The bandwidths-weights line appears At Most Once for a consensus. It does
not appear in votes.
```

"bandwidth-weights" [SP Weights] NL

```
Weight ::= Keyword '=' Int32
Int32 ::= A decimal integer between -2147483648 and 2147483647.
Weights ::= Weight | Weights SP Weight
```

List of optional weights to apply to router bandwidths during path selection. They are sorted in lexical order (as ASCII byte strings) 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.

The signature contains the following item, which appears Exactly Once for a vote, and At Least Once for a consensus.

"directory-signature" [SP Algorithm] SP identity SP signing-key-digest NL Signature

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.)

3.4.2. 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" -- Directory authority administrators may decide to support name binding. If they do, then they must maintain a file of nickname-to-identity-key mappings, and try to keep this file consistent with other directory authorities. If they don't, they act as clients, and report bindings made by other directory authorities (name X is bound to identity Y if at least one binding directory lists it, and no directory binds X to some other Y'.) A router is called 'Named' if the router believes the given name should be bound to the given key.

Two strategies exist on the current network for deciding on values for the Named flag. In the original version, relay operators were asked to send nickname-identity pairs to a mailing list of Naming directory authorities' operators. The operators were then supposed to add the pairs to their mapping files; in practice, they didn't get to this often.

Newer Naming authorities run a script that registers routers in their mapping files once the routers have been online at least two weeks, no other router has that nickname, and no other router has wanted the nickname for a month. If a router has not been online for six months, the router is removed.

"Unnamed" -- Directory authorities that support naming should vote for a router to be 'Unnamed' if its given nickname is mapped to a different identity.

"Running" -- A router is 'Running' if the authority managed to connect to it successfully within the last 45 minutes.

"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 α^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 two of the ports 80, 443, and 6667 and allows exits to at least one /8 address space.

"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 some minimum (20KB/s until 0.2.3.7-alpha, and 100KB/s after that).

"Guard" -- A router is a possible 'Guard' if its Weighted Fractional Uptime is at least the median for "familiar" active routers, and if its bandwidth is at least median or at least 250KB/s.

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.

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, and it is running a version of the directory protocol that supports the functionality clients need. (Currently, this is 0.1.1.9-alpha or later.)

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

Directory server administrators may label some relays or IPs as blacklisted, and elect not to include them in their network-status lists.

Authorities SHOULD 'disable' any relays in excess of 2 on any single IP. When there are more than 2 (or AuthDirMaxServersPerAddr) 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[*]. To 'disable' a relay, the authority *should* advertise it without the Running or Valid flag.

[*] In this comparison, measured bandwidth is used unless it is not present for a router, in which case advertised bandwidth is used.

Bug 8710 has a patch to change this behavior.

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.

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.

3.5. Downloading missing certificates from other directory authorities

XXX when to download certificates.

3.6. 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]

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

3.7. 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.

3.8. Computing a consensus from a set of votes

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.z
```

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.

```
     Consensus methods 11 and before, entries are 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 method 19 or later, 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.

     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 21 or later)

        * For any given RSA or 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.)

        * 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.

        * 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, votes and
          consensuses may include "a" lines listing additional OR
          ports.

        * 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.

     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.

3.8.0.1. Deciding which Ids to include.

  For each <id-Ed, id-RSA> that is listed by more than half of the total
    authorities (not just total votes), include it.  (No other <id-Ed, id-RSA'>
    can have as many votes.)

  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 without an id-Ed.

3.8.0.2 Deciding which descriptors to include

   Deciding which descriptors to include.

   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 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.


3.8.1. 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.
     "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 and to voting
             algorithm.
```

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).

All authorities MUST support method 13; authorities SHOULD support more recent methods as well. Authorities SHOULD NOT support or advertise support for any method before 13.

(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.)

3.8.2. Encoding port lists

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.]

3.8.3. Computing Bandwidth Weights

Let weight_scale = 10000

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 = 1)
    Wmg*G + Wgg*G == G                          (aka: Wgg = 1-Wmg)
    Wme*E + Wee*E == E                          (aka: Wee = 1-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 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 = 1, 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 == 1 and Wee == 1, since both those positions are scarce:

```
         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 = 1, 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 == 1, 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).

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
```

3.9. 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.

Every consensus flavor has a name consisting of a sequence of one or more alphanumeric characters and dashes. For compatibility, current descriptor flavor 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
```

3.9.1. ns consensus

The ns consensus flavor is equivalent to the unflavored consensus except for its first line which states its consensus flavor name:

```
 "network-status-version" SP version SP "ns" NL

     [At start, exactly once.]
```

3.9.2. Microdescriptor consensus

network-status-microdesc-consensus-3 1.0

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.

 "p" ... NL

     [Zero times.]

     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 MAY use the sha256 digest algorithm for its signatures.

3.10. Exchanging detached signatures

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.]

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

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

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.]

3.11. Publishing the signed consensus

```
   Once there are enough signatures, or once the voting period starts,
   these documents are available at
      http://<hostname>/tor/status-vote/current/consensus.z
   and
      http://<hostname>/tor/status-vote/current/consensus-signatures.z
   [XXX current/consensus-signatures is not currently implemented, as it
    is not used in the voting protocol.]
   [XXX It's actually false that the first document is available as soon
    as there are enough signatures. It's only available as soon as the
    voting period starts. -KL]

   [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.z
     http://<hostname>/tor/status-vote/current/<fp>.z
     http://<hostname>/tor/status-vote/current/d/<d>.z
   once the consensus is complete.

   The authorities serve another consensus of each flavor "F" from the
   locations
      /tor/status-vote/(current|next)/consensus-F.z. and
      /tor/status-vote/(current|next)/consensus-F/<FP1>+....z.
```

4. Directory cache operation

```
   All directory caches implement this section, except as noted.
```

4.1. 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.
```

4.2. 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]
```

4.3. Downloading microdescriptors from directory authorities

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

   The microdescriptors with base64 hashes <D1>,<D2>,<D3> are available
   at:
     http://<hostname>/tor/micro/d/<D1>-<D2>-<D3>[.z]

   <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 microdescrriptor hashes
can be retrieved in a single request.)

4.4. 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.

5. Client operation

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

5.1. Downloading network-status documents

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

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-until field
has not passed.

When a client has no consensus network-status document, it downloads it
from a randomly chosen authority. 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; see 6 below.)

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).

[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, they should not need to go to the authority directly
again.)

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 cache has a consensus that became valid at 1:00,
and is fresh until 2:00, and expires at 4:00, that cache 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.

5.2. 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 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 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 not too old (currently, 48 hours), OR so long as no better descriptor has been downloaded for the same router.

[Versions of Tor before 0.1.2.3-alpha would discard descriptors simply for being published too far in the past.] [The code seems to discard descriptors in all cases after they're 5 days old. True? -RD]

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. If it needs to learn more than half of the microdescriptors, it requests 'all', else it requests only the missing ones. Clients MAY try to determine whether the upload bandwidth for listing the microdescriptors they want is more or less than the download bandwidth for the microdescriptors they do not want.
[XXX The 'all' URL is not implemented yet. -KL]

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.

5.3. 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.

5.4. Using directory information

[XXX This subsection really belongs in path-spec.txt, not here. -KL]

Everyone besides directory authorities uses the approaches in this section to decide which relays to use and what their keys are likely to be. (Directory authorities just believe their own opinions, as in section 3.4.2 above.)

5.4.1. Choosing routers for circuits.

Circuits SHOULD NOT be built until the client has enough directory information: a live consensus network status [XXXX fallback?] and descriptors for at least 1/4 of the relays believed to be running.

A relay is "listed" if it is included by the consensus network-status document. Clients SHOULD NOT use unlisted relays.

These flags are used as follows:

- Clients SHOULD NOT use non-'Valid' or non-'Running' routers unless requested to do so.

- Clients SHOULD NOT use non-'Fast' routers for any purpose other than very-low-bandwidth circuits (such as introduction circuits).

- Clients SHOULD NOT use non-'Stable' routers for circuits that are likely to need to be open for a very long time (such as those used for IRC or SSH connections).

- Clients SHOULD NOT choose non-'Guard' nodes when picking entry guard nodes.

See the "path-spec.txt" document for more details.

5.4.2. Managing naming

In order to provide human-memorable names for individual router identities, some directory servers bind names to IDs. Clients handle names in two ways:

When a client encounters a name it has not mapped before:

> If the consensus lists any router with that name as "Named", or if consensus-method 2 or later is in use and the consensus lists any router with that name as having the "Unnamed" flag, then the name is bound. (It's bound to the ID listed in the entry with the Named, or to an unknown ID if no name is found.)

When the user refers to a bound name, the implementation SHOULD provide only the router with ID bound to that name, and no other router, even if the router with the right ID can't be found.

When a user tries to refer to a non-bound name, the implementation SHOULD warn the user. After warning the user, the implementation MAY use any router that advertises the name.

Not every router needs a nickname. When a router doesn't configure a nickname, it publishes with the default nickname "Unnamed". Authorities SHOULD NOT ever mark a router with this nickname as Named; client software SHOULD NOT ever use a router in response to a user request for a router called "Unnamed".

5.4.3. Software versions

An implementation of Tor SHOULD warn when it has fetched a consensus network-status, and it is running a software version not listed.

5.4.4. Warning about a router's status.

If a router tries to publish its descriptor to a Naming authority that has its nickname mapped to another key, the router SHOULD warn the operator that it is either using the wrong key or is using an already claimed nickname.

If a router has fetched a consensus document,, and the authorities do not publish a binding for the router's nickname, the router MAY remind the operator that the chosen nickname is not bound to this key at the authorities, and suggest contacting the authority operators.

...

5.4.5. Router protocol versions

A client should believe that a router supports a given feature if that feature is supported by the router or protocol versions in more than half of the live networkstatuses' "v" entries for that router. In other words, if the "v" entries for some router are:

```
    v Tor 0.0.8pre1                (from authority 1)
    v Tor 0.1.2.11                 (from authority 2)
    v FutureProtocolDescription 99 (from authority 3)
```

then the client should believe that the router supports any feature supported by 0.1.2.11.

This is currently equivalent to believing the median declared version for a router in all live networkstatuses.

6. Standards compliance

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

6.1. HTTP headers

Servers MAY set the Content-Length: header. Servers SHOULD set Content-Encoding to "deflate" or "identity".

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.

6.2. 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.

```
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.
```

A. 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
```

B. General-use HTTP URLs

"Fingerprints" in these URLs are base16-encoded SHA1 hashes.

The most recent v3 consensus should be available at:

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

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

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

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

Where F1, F2, etc. are authority identity fingerprints 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.

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.

A concatenated set of all the current key certificates should be available at:

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

The key certificate for this server (if it is an authority) should be available at:

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

The key certificate for an authority whose authority identity fingerprint is <F> should be available at:

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

The key certificate whose signing key fingerprint is <F> should be available at:

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

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

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

(As usual, clients may request multiple certificates using:
    http://<hostname>/tor/keys/fp-sk/<F1>-<S1>+<F2>-<S2>.z)
[The above fp-sk format was not supported before Tor 0.2.1.9-alpha.]

The most recent descriptor for a server whose identity key has a fingerprint of <F> should be available at:
    http://<hostname>/tor/server/fp/<F>.z

The most recent descriptors for servers with identity fingerprints <F1>,<F2>,<F3> should be available at:
    http://<hostname>/tor/server/fp/<F1>+<F2>+<F3>.z

(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 (descriptor) digest <D> (in hex) should be available at:
    http://<hostname>/tor/server/d/<D>.z

The most recent descriptors with digests <D1>,<D2>,<D3> should be available at:
    http://<hostname>/tor/server/d/<D1>+<D2>+<D3>.z

The most recent descriptor for this server should be at:
    http://<hostname>/tor/server/authority.z
 [Nothing in the Tor protocol uses this resource yet, but it is useful for debugging purposes. Also, the official Tor implementations (starting at 0.1.1.x) use this resource to test whether a server's own DirPort is reachable.]

A concatenated set of the most recent descriptors for all known servers should be available at:
    http://<hostname>/tor/server/all.z

Extra-info documents are available at the URLS
    http://<hostname>/tor/extra/d/...
    http://<hostname>/tor/extra/fp/...
    http://<hostname>/tor/extra/all[.z]
    http://<hostname>/tor/extra/authority[.z]
        (As for /tor/server/ URLs: supports fetching extra-info documents by their 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".)

For debugging, directories SHOULD expose non-compressed objects at URLs like the above, but without the final ".z".
Clients MUST handle compressed concatenated information in two forms:
  - A concatenated list of zlib-compressed objects.
  - A zlib-compressed concatenated list of objects.
Directory servers MAY generate either format: the former requires less CPU, but the latter requires less bandwidth.

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

[XXX Add new URLs for microdescriptors, consensus flavors, and microdescriptor consensus. -KL]

C. Converting a curve25519 public key to an ed25519 public key

Given a curve25519 x-coordinate (u), we can get the y coordinate of the ed25519 key using

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

and then we can apply the usual ed25519 point decompression algorithm to find the x coordinate of the ed25519 point to check signatures with.

Note that 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 key. Therefore, we need the 'sign' of the X coordinate, as used by the ed25519 key expansion algorithm.

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