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

Commonly used *ThresholdKey*s 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.

`required-` / `recommended-*-protocols` — Minimum protocol features

- `recommended-client-protocols` *entry entry* ..
- `recommended-relay-protocols` *entry entry* ..
- `required-client-protocols` *entry entry* ..
- `required-relay-protocols` *entry entry* ..
- At most once each.
- Zero or more *entry* arguments.

entry is as for `proto` in a server descriptor.

To vote on these entries, a protocol/version combination is included only if it is listed by a majority of the voters.

Directory servers may be:

- authoritative directories (RELAY_BEGIN_DIR, usually non-anonymous),
- bridge authoritative directories (RELAY_BEGIN_DIR, anonymous),
- directory mirrors (RELAY_BEGIN_DIR, usually non-anonymous),
- onion service directories (RELAY_BEGIN_DIR, anonymous).

If the Tor relay is not running a directory service, it should respond with a REASON_NOTDIRECTORY RELAY_END message.

Clients MUST generate a empty body for RELAY_BEGIN_DIR message; relays MUST ignore the the body of a RELAY_BEGIN_DIR message.

In response to a RELAY_BEGIN_DIR message, relays respond either with a RELAY_CONNECTED message on success, or a RELAY_END message on failure. They MUST send a RELAY_CONNECTED message with an empty body; clients MUST ignore the body.

Closing streams

When an anonymized TCP connection is closed, or an edge node encounters error on any stream, it sends a 'RELAY_END' message along the circuit (if possible) and closes the TCP connection immediately. If an edge node receives a 'RELAY_END' message for any stream, it closes the TCP connection completely, and sends nothing more along the circuit for that stream.

The body of a RELAY_END message begins with a single 'reason' byte to describe why the stream is closing. For some reasons, it contains additional data (depending on the reason.) The values are:

```
       1 -- REASON_MISC           (catch-all for unlisted reasons)
       2 -- REASON_RESOLVEFAILED  (couldn't look up hostname)
       3 -- REASON_CONNECTREFUSED (remote host refused connection) [*]
       4 -- REASON_EXITPOLICY     (Relay refuses to connect to host or
port)
       5 -- REASON_DESTROY        (Circuit is being destroyed)
       6 -- REASON_DONE           (Anonymized TCP connection was
closed)
       7 -- REASON_TIMEOUT        (Connection timed out, or relay
timed out
                                   while connecting)
       8 -- REASON_NOROUTE        (Routing error while attempting to
                                   contact destination)
       9 -- REASON_HIBERNATING    (Relay is temporarily hibernating)
      10 -- REASON_INTERNAL       (Internal error at the relay)
      11 -- REASON_RESOURCELIMIT  (Relay has no resources to fulfill
request)
      12 -- REASON_CONNRESET      (Connection was unexpectedly reset)
      13 -- REASON_TORPROTOCOL    (Sent when closing connection
because of
                                   Tor protocol violations.)
      14 -- REASON_NOTDIRECTORY   (Client sent RELAY_BEGIN_DIR to a
                                   non-directory relay.)

   [*] Older versions of Tor also send this reason when connections
are
       reset.
```

Clients and relays MUST accept reasons not on the above list, since future versions of Tor may provide more fine-grained reasons.

For most reasons, the format of RELAY_END is:

Reason [1 byte]

```
        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-space printing
characters
              other than "=".
```

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 methods 19-33. Earlier methods did not include this; method 34 removed it.

`known-flags` — Router flags which could be determined

- `known-flags` *flag flag* ..
- Exactly once.
- One or more distinct `_flag_` arguments, in lexical order

A space-separated list of all of the flags that this document might contain in `s` Items.

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.

`flag-thresholds` — Authority's performance thresholds

- `flag-thresholds` *threshold threshold* ..
- One or more *threshold* arguments
- At most once for votes; does not occur in consensuses.

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

In practice, clients continue to use the consensus for up to 24 hours after it is no longer valid, if no more recent consensus can be downloaded.

See Voting timeline.

`voting-delay` — Vote timing parameters

- `voting-delay` *VoteSeconds DistSeconds* ..
- 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 Voting timeline.

`client-versions` — Recommended Tor client software

- `client-versions` *version*,*version*,.. ..
- At most once

A comma-separated list of recommended Tor ***version***s 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` — Recommended Tor server software

- `server-versions` *version*,*version*,.. ..
- 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` — Software distribution hashes (obsolete)

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

[Any number of times.]

For this element:

For REASON_EXITPOLICY, the format of RELAY_END is:

```
Reason                      [1 byte]
IPv4 or IPv6 address        [4 bytes or 16 bytes]
TTL                         [4 bytes]
```

(If the TTL is absent, it should be treated as if it were 0xffffffff. If the address is absent or is the wrong length, the RELAY_END message should be processed anyway.)

Tors SHOULD NOT send any reason except REASON_MISC for a stream that they have originated.

Implementations SHOULD accept empty RELAY_END messages, and treat them as if they specified REASON_MISC.

Upon receiving a RELAY_END message, the recipient may be sure that no further messages will arrive on that stream, and can treat such messages as a protocol violation.

Upon receiving a RELAY_END message, the recipient MAY respond with a RELAY_END message with the reason set to REASON_MISC.

> Note: as of 2025 current implementations do not automatically send RELAY_END after receiving RELAY_END.

After sending a RELAY_END message, the sender needs to give the recipient time to receive that message. In the meantime, the sender SHOULD remember how many messages of which types (CONNECTED, SENDME, DATA) it would have accepted on that stream, and SHOULD kill the circuit if it receives more than permitted.

— [The rest of this section describes unimplemented functionality.]

Because TCP connections can be half-open, we follow an equivalent to TCP's FIN/FIN-ACK/ACK protocol to close streams.

An exit (or onion service) connection can have a TCP stream in one of three states: 'OPEN', 'DONE_PACKAGING', and 'DONE_DELIVERING'. For the purposes of modeling transitions, we treat 'CLOSED' as a fourth state, although connections in this state are not, in fact, tracked by the onion router.

A stream begins in the 'OPEN' state. Upon receiving a 'FIN' from the corresponding TCP connection, the edge node sends a 'RELAY_FIN' message along the circuit and changes its state to 'DONE_PACKAGING'. Upon receiving a 'RELAY_FIN' message, an

edge node sends a 'FIN' to the corresponding TCP connection (e.g., by calling shutdown(SHUT_WR)) and changing its state to 'DONE_DELIVERING'.

When a stream in already in 'DONE_DELIVERING' receives a 'FIN', it also sends a 'RELAY_FIN' along the circuit, and changes its state to 'CLOSED'. When a stream already in 'DONE_PACKAGING' receives a 'RELAY_FIN' message, it sends a 'FIN' and changes its state to 'CLOSED'.

If an edge node encounters an error on any stream, it sends a 'RELAY_END' message (if possible) and closes the stream immediately.

`published` — Publication time

- **`published` *date time* ..
- *date time* is *YYYY-MM-DD HH:MM:SS* and is in UTC
- Exactly once
- In votes only

The publication time for this vote.

See Voting timeline.

`valid-after` — Start of the Interval

- `valid-after` *date time* ..
- *date time* is *YYYY-MM-DD HH:MM:SS* and is in UTC
- Exactly once

The start of the Interval for this vote. Before this time, the consensus document produced from this vote is not officially in use.

(Note that because of propagation delays, clients and relays may see consensus documents that are up to `DistSeconds` earlier than this time, and should not warn about them.)

See Voting timeline.

`fresh-until` — Time until no longer fresh

- `fresh-until` *date time* ..
- *date time* is *YYYY-MM-DD HH:MM:SS* and is in UTC
- 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 Voting timeline.

`valid-until` — Time until no longer valid

- `valid-until` *date time* ..
- *date time* is *YYYY-MM-DD HH:MM:SS* and is in UTC
- Exactly once

The end of the Interval for this vote. After this time, all clients should try to find a more recent consensus.

These rules MUST be followed by a dirauth when generating a consensus. When reading a vote or consensus, an implementation MUST NOT insist on them.

> When reading, the usual relaxed parsing is needed to support future compatibility. When writing, `consensus-methods` in votes allows the participating dirauths to negotiate protocol upgrades.

Vote and consensus document, preamble items

The preamble contains the following items.

`network-status-version` — Document format version

- `network-status-version` version ..
- At start, exactly once.

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

`vote-status` — Declare document type

- `vote-status` type ..
- Exactly once.
- ***type*** is `vote` or `status`

`consensus-methods` – Supported consensus methods

- `consensus-methods` *method method* ..
- At most once
- In votes only
- At least one *method* argument.

The consensus methods supported by this voter. Each ***method*** is a as a decimal integer. See Computing a consensus for details. Absence of the line means that only method "1" is supported.

`consensus-method` — Consensus method used

- `consensus-method` *method*
- At most once
- In consensus only

See Computing a consensus for details.

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

Remote hostname lookup

To find the address associated with a hostname, the client sends a RELAY_RESOLVE message containing the hostname to be resolved with a NUL terminating byte.

For a reverse lookup, the client sends a RELAY_RESOLVE message containing an in-addr.arpa address.

The relay replies with a RELAY_RESOLVED message containing any number of answers. Each answer is of the form:

```
      Type   (1 octet)
      Length (1 octet)
      Value  (variable-width)
      TTL    (4 octets)
   "Length" is the length of the Value field.
   "Type" is one of:

      0x00 -- Hostname
      0x04 -- IPv4 address
      0x06 -- IPv6 address
      0xF0 -- Error, transient
      0xF1 -- Error, nontransient
```

If any answer has a type of 'Error', then no other answer may be given.

The 'Value' field encodes the answer:

- IP addresses are given in network order.
- Hostnames are given in standard DNS order ("www.example.com") and not NUL-terminated.
- The content of Errors is currently ignored. Relays currently set it to the string "Error resolving hostname" with no terminating NUL. Implementations MUST ignore this value.

For backward compatibility, if there are any IPv4 answers, one of those must be given as the first answer.

The RELAY_RESOLVE messge must use a nonzero, distinct streamID; the corresponding RELAY_RESOLVED message must use the same streamID. No stream is actually created by the relay when resolving the name.

Flow control

Link throttling

Each client or relay should do appropriate bandwidth throttling to keep its user happy.

Communicants rely on TCP's default flow control to push back when they stop reading.

The mainline Tor implementation uses token buckets (one for reads, one for writes) for the rate limiting.

Since 0.2.0.x, Tor has let the user specify an additional pair of token buckets for "relayed" traffic, so people can deploy a Tor relay with strict rate limiting, but also use the same Tor as a client. To avoid partitioning concerns we combine both classes of traffic over a given relay connection, and keep track of the last time we read or wrote a high-priority (non-relayed) cell. If it's been less than N seconds (currently N=30), we give the whole connection high priority, else we give the whole connection low priority. We also give low priority to reads and writes for connections that are serving directory information. See proposal 111 for details.

Link padding

Link padding can be created by sending PADDING or VPADDING cells along the connection; relay messages of type "DROP" can be used for long-range padding. The bodies of PADDING cells, VPADDING cells, or DROP message are filled with padding bytes. See Cell Packet format.

If the link protocol is version 5 or higher, link level padding is enabled as per padding-spec.txt. On these connections, clients may negotiate the use of padding with a PADDING_NEGOTIATE command whose format is as follows:

```
Version          [1 byte]
Command          [1 byte]
ito_low_ms       [2 bytes]
ito_high_ms      [2 bytes]
```

Vote and consensus status document formats

Each status document contains (in this order):

- a preamble
- an authority section
- zero or more router status entries
- a footer
- one or more signatures.

Some items appear only in votes, and some items appear only in consensuses. Unless specified, items occur in both. Items appearing only in certain documents is indicated by one of the following in the Item's syntax bullet points:

- In votes only
- In consensus only

Consensus documents may also be available in "flavored" forms computed (deterministically) from the information desccribed here.

> Generation rules for consensus items are, partly, specified separately. and partly here. Refer to both places.

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

Consensus documents — reproducibility

Consensus documents generated by each dirauth must be byte-for-byte identical, as the same document must be countersigned by multiple dirauths.

Therefore, when generating a consensus document, certain of the metaformat rules are tightened:

- Each WS is precisely a single SP.
- Extra arguments are not allowed.
- Items (and where applicable, arguments) are ordered as presented/specified here.

Also, once an authority receives a vote from another authority, it examines it for new descriptors and fetches them from that authority. This may be 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.

Currently, only version 0 of this cell is defined. In it, the command field is either 1 (stop padding) or 2 (start padding). For the start padding command, a pair of timeout values specifying a low and a high range bounds for randomized padding timeouts may be specified as unsigned integer values in milliseconds. The ito_low_ms field should not be lower than the current consensus parameter value for nf_ito_low (default: 1500). The ito_high_ms field should not be lower than ito_low_ms. (If any party receives an out-of-range value, they clamp it so that it is in-range.)

For the stop padding command, the timeout fields should be sent as zero (to avoid client distinguishability) and ignored by the recipient.

For more details on padding behavior, see padding-spec.txt.

Circuit-level flow control

To control a circuit's bandwidth usage, each relay keeps track of two 'windows', consisting of how many DATA-bearing relay cells it is allowed to originate or willing to consume.

(For the purposes of flow control, we call a relay cell "DATA-bearing" if it holds a DATA relay message. Note that this design does *not* limit relay cells that don't contain a DATA message; this limitation may be addressed in the future.)

These two windows are respectively named: the package window (packaged for transmission) and the deliver window (delivered for local streams).

Because of our leaky-pipe topology, every relay on the circuit has a pair of windows, and the client has a pair of windows for every relay on the circuit. These windows apply only to *originated* and *consumed* cells. They do not, however, apply to *relayed* cells, and a relay that is never used for streams will never decrement its windows or cause the client to decrement a window.

Each 'window' value is initially set based on the consensus parameter 'circwindow' in the directory (see dir-spec.txt), or to 1000 DATA-bearing relay cells if no 'circwindow' value is given. In each direction, cells that are not RELAY_DATA cells do not affect the window.

A relay or client (depending on the stream direction) sends a RELAY_SENDME message to indicate that it is willing to receive more DATA-bearing cells when its deliver window goes down below a full increment (100). For example, if the window started at 1000, it should send a RELAY_SENDME when it reaches 900.

When a relay or client receives a RELAY_SENDME, it increments its package window by a value of 100 (circuit window increment) and proceeds to sending the remaining DATA-bearing cells.

If a package window reaches 0, the relay or client stops reading from TCP connections for all streams on the corresponding circuit, and sends no more DATA-bearing cells until receiving a RELAY_SENDME message.

If a deliver window goes below 0, the circuit should be torn down.

Starting with tor-0.4.1.1-alpha, authenticated SENDMEs are supported (version 1, see below). This means that both the relay and client need to remember the rolling digest of the relay cell that precedes (triggers) a RELAY_SENDME. This can be known if the package window gets to a multiple of the circuit window increment (100).

When the RELAY_SENDME version 1 arrives, it will contain a digest that MUST match the one remembered. This represents a proof that the end point of the circuit saw the sent relay cells. On failure to match, the circuit should be torn down.

To ensure unpredictability, random bytes should be added to at least one RELAY_DATA cell within one increment window. In other word, at every 100 data-bearing cells (increment), random bytes should be introduced in at least one cell.

SENDME Message Format

A circuit-level RELAY_SENDME message always has its StreamID=0.

A relay or client must obey these two consensus parameters in order to know which version to emit and accept.

```
'sendme_emit_min_version': Minimum version to emit.
'sendme_accept_min_version': Minimum version to accept.
```

If a RELAY_SENDME version is received that is below the minimum accepted version, the circuit should be closed.

The body of a RELAY_SENDME message contains the following:

Field	Size in bytes
VERSION	1
DATA_LEN	2

Exchanging votes

> Note that all of the HTTP GET URLs here have variants ending in ".z". The URLs for HTTP POST do not.

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`

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>`

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

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

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

“id” SP “rsa1024” SP base64-encoded-identity-digest NL

[At most once]

The node identity digest `SHA1(DER(KP_relayid_rsa))`, 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.

All implementations MUST ignore this key for any microdescriptor whose corresponding entry in the consensus includes the ‘NoEdConsensus’ flag.

(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.)

Field	Size in bytes
`DATA`	`DATA_LEN`

The VERSION tells us what is expected in the DATA section of length DATA_LEN and how to handle it. The recognized values are:

- 0x00: The rest of the message should be ignored.
- 0x01: Authenticated SENDME. The DATA section MUST contain:

  DIGEST [20 bytes]

  If the DATA_LEN value is less than 20 bytes, the message should be dropped and the circuit closed. If the value is more than 20 bytes, then the first 20 bytes should be read to get the DIGEST value.

  The DIGEST is the rolling digest value from the DATA-bearing relay cell that immediately preceded (triggered) this RELAY_SENDME. This value is matched on the other side from the previous cell sent that the relay/client must remember.

  (Note that if the digest in use has an output length greater than 20 bytes—as is the case for the hop of an onion service rendezvous circuit created by the hs_ntor handshake—we truncate the digest to 20 bytes here.)

If the VERSION is unrecognized or below the minimum accepted version (taken from the consensus), the circuit should be torn down.

Stream-level flow control

Edge nodes use RELAY_SENDME messages to implement end-to-end flow control for individual connections across circuits. Similarly to circuit-level flow control, edge nodes begin with a window of DATA-bearing cells (500) per stream, and increment the window by a fixed value (50) upon receiving a RELAY_SENDME message. Edge nodes initiate RELAY_SENDME messages when both a) the window is <= 450, and b) there are less than ten cells’ worth of data remaining to be flushed at that edge.

Stream-level RELAY_SENDME messages are distinguished by having nonzero StreamID. They are still empty; the body still SHOULD be ignored.

Subprotocol-based versioning

Tor implementations use “subprotocol versioning” to describe and negotiate which versions and features of the Tor protocol they support.

Individual protocol features (which we will call “subprotocol capabilities”) are grouped by the area of the protocol to which they apply, and denoted by individual numbers.

> For example, “Relay=3” is a subprotocol capability, as is “Link=5”.

Each subprotocol capability also has a symbolic name for use by programmers. These names are for human convenience only, and not used within the Tor protocols.

> For example, “Relay=3” is also known as `RELAY_EXTEND_IPV6`.

Relays advertise their supported subprotocol capabilities in the “proto” field in their descriptors. Authorities re-publish this information in the “pr” field of the relay’s microdescriptor.

Recognized protocols are as follows. When we need to indicate a protocol numerically in our protocol, we use the numeric Ids in this table.

Protocol	Numeric Id
Link	0
LinkAuth	1
Relay	2
DirCache	3
HSDir	4
HSIntro	5
HSRend	6
Desc	7
Microdesc	8
Cons	9
Padding	10
FlowCtrl	11

(Note that if an entry is not of the form “nickname”, “$hexid”, “$hexid=nickname” or “$hexid~nickname”, then it will be unchanged: this is what makes the algorithm forward-compatible.)

Clients use these family lists to determine family membership when building paths.

"family-ids" SP ids SP NL

[At most once.]

ids is a space-separated list of *Family IDs*. Each family ID consists of any number of otherwise valid nonspace characters.

Authorities generate a `family-ids` entry by deriving an ID from each of the `family-certs` listed in the relay’s router descriptor, sorting those IDs in lexicographic order, and removing any duplicates.

Clients use these family IDs to determine family membership when building paths.

Clients SHOULD accept family IDs in unrecognized formats.

[This entry first appeared in consensus method 35. Earlier methods should omit it.]

“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.

[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.]

[In consensus methods before 5, this line was omitted.]

“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.)

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

[Before Tor 0.4.5.1-alpha, this field was optional.]

“a” SP address “:” port NL

[Any number]

Additional advertised addresses for the OR.

Present currently only if the OR advertises at least one IPv6 address; currently, the first address is included and all others are omitted. Any other IPv4 or IPv6 addresses should be ignored.

Address and port are as for “or-address” as specified in section 2.1.1.

(Only included when generating microdescriptors for consensus-methods 14 to 27.)

"family" names NL

[At most once]

The “family” element as specified in server descriptors.

When generating microdescriptors for consensus method 29 or later, the following canonicalization algorithm is applied to improve compression:

```
For all entries of the form $hexid=name or $hexid~name,
remove the =name or ~name portion.

Remove all entries of the form $hexid, where hexid is not
40 hexadecimal characters long.

If an entry is a valid nickname, put it into lower case.

If an entry is a valid $hexid, put it into upper case.

If there are any entries, add a single $hexid entry for
the relay in question, so that it is a member of its own
family.

Sort all entries in lexical order.

Remove duplicate entries.
```

Protocol	Numeric Id
Conflux	12

Interpreting subprotocol capabilities

Each subprotocol capability should be interpreted as a single flag, independent of all others.

That is to say, from the fact that an instance supports “Relay=5”, it is incorrect to conclude that the relay supports “Relay=4”, even though 5 is greater than 4.

> Earlier versions of this document, and some other places in our spec and code, refer to “subprotocol versions”. We are avoiding this vocabulary in the future, to avoid this misunderstanding.

Required and recommended subprotocols

The consensus document contains lists of subprotocol capabilities that are recommend or required for relays and clients.

They are:

- “recommended-client-protocols”
- “recommended-relay-protocols”
- “required-relay-protocols”
- “required-client-protocols”

Here are the rules a relay and client should follow when encountering a protocol list in the consensus:

- When a relay lacks a capability listed in recommended-relay-protocols, it should warn its operator that the relay is obsolete.
- When a relay lacks a capability listed in required-relay-protocols, it should warn its operator as above. If the consensus is newer than the date when the software was released or scheduled for release, it must not attempt to join the network.
- When a client lacks a capability listed in recommended-client-protocols, it should warn the user that the client is obsolete.
- When a client lacks a capability listed in required-client-protocols, it should warn the user as above. If the consensus is newer than the date when the

software was released, it must not connect to the network. This implements a “safe forward shutdown” mechanism for zombie clients.

- If a client or relay has a cached consensus telling it that a given capability is required, and it does not implement that capability, it SHOULD NOT try to fetch a newer consensus.

Software release dates SHOULD be automatically updated as part of the release process, to prevent forgetting to move them forward. Software release dates MAY be manually adjusted by maintainers if necessary.

Starting in version 0.2.9.4-alpha, the initial required subprotocol capabilities for clients that we will Recommend and Require are:

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

For relays we will Require:

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

For relays, we will additionally Recommend all subprotocol capabilities which we recommend for clients.

“Link”

The “link” protocols are those used by clients and relays to initiate and receive relay connections and to handle cells on relay connections. The “link” subprotocols correspond 1:1 to those versions.

Two Tor instances can make a connection to each other only if they have at least one link protocol in common.

The current “link” capabilities are: “1” through “5”. See Negotiating versions with VERSIONS cells for more information.

“LinkAuth”

LinkAuth protocols correspond to varieties of AUTHENTICATE cells used for the v3+ link protocols.

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. A microdescriptor is identified by the SHA256 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 (Optionally, a public key in PEM format)

[Exactly once, at start] [No extra arguments]

Optionally, an obsolete TAP key. (Note that while the *key* is optional, the `onion-key` element itself is not. It is used to denote the start of a microdescriptor.)

This key is no longer used for anything. If present, it MUST be in the same format as described for relay descriptors.

All current consensus methods generate a key in this position.

“ntor-onion-key” SP base-64-encoded-key NL

[Exactly once]

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

When generating microdescriptors for consensus method 30 or later, the trailing = sign must be absent. For consensus method 29 or earlier, the trailing = sign must be present.

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.

Current subprotocol capabilities are:

- "1" (`LINKAUTH_RSA_SHA256_TLSSECRET`) – the RSA link authentication described in Link authentication type 1: RSA-SHA256-TLSSecret.
- "2" is unused, and reserved by proposal 244.
- "3" (`LINKAUTH_ED25519_SHA256_EXPORTER`) – the ed25519 link authentication described in Link authentication type 3: Ed25519-SHA256-RFC5705.

"Relay"

The "relay" protocols are those used to handle CREATE/CREATE2 cells, and those that handle the various relay messages received after a CREATE/CREATE2 cell. (Except, relay cells used to manage introduction and rendezvous points are managed with the "HSIntro" and "HSRend" protocols respectively.)

Current subprotocol capabilities are as follows.

- "1" (`RELAY_BASE`) – supports the TAP key exchange, with all features in Tor 0.2.3. Support for CREATE and CREATED.
- "2" (`RELAY_NTOR`) – supports the ntor key exchange, and all features in Tor 0.2.4.19. Includes support for CREATE2 and CREATED2 and EXTEND2 and EXTENDED2.

  Relay=2 has limited IPv6 support:

  - Clients might not include IPv6 ORPorts in EXTEND2 messages.
  - Relays (and bridges) might not initiate IPv6 connections in response to EXTEND2 messages containing IPv6 ORPorts, even if they are configured with an IPv6 ORPort.

  However, relays support accepting inbound connections to their IPv6 ORPorts. And they might extend circuits via authenticated IPv6 connections to other relays.
- "3" (`RELAY_EXTEND_IPv6`) – relays support extending over IPv6 connections in response to an EXTEND2 message containing an IPv6 ORPort.

  Bridges might not extend over IPv6, because they try to imitate client behaviour.

A successful IPv6 extend requires:

- Relay=3 subprotocol (or later) on the extending relay,
- an IPv6 ORPort on the extending relay,
- an IPv6 ORPort for the accepting relay in the EXTEND2 message, and
- an IPv6 ORPort on the accepting relay. (Because different tor instances can have different views of the network, these checks should be done when the path is selected. Extending relays should only check local IPv6 information, before attempting the extend.)

When relays receive an EXTEND2 message containing both an IPv4 and an IPv6 ORPort, and there is no existing authenticated connection with the target relay, the extending relay may choose between IPv4 and IPv6 at random. The extending relay might not try the other address, if the first connection fails.

As is the case with other subprotocols, tor advertises, recommends, or requires support for this subprotocol, regardless of its current configuration.

In particular:

- relays without an IPv6 ORPort, and
- tor instances that are not relays, have the following behaviour, regardless of their configuration:
- advertise support for "Relay=3" in their descriptor (if they are a relay, bridge, or directory authority), and
- react to consensuses recommending or requiring support for "Relay=3".

This subprotocol is described in proposal 311, and implemented in Tor 0.4.5.1-alpha.

- "4" (`RELAY_NTORV3`) – support the ntorv3 (version 3) key exchange and all features in 0.4.7.3-alpha. This adds a new CREATE2 cell type. See proposal 332 and The "ntor-v3" handshake for more details.

- "5" (`RELAY_NEGOTIATE_SUBPROTO`) – support the ntorv3 subprotocol request extension (proposal 346) allowing a client to request what features to be used on a circuit.

- "6" (`RELAY_CRYPT_CGO`) [RESERVED] – Support the Counter Galois Onion relay encryption algorithm (proposal 359).

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 an 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, 2 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.

`dir-key-crosscert` — Cross-certificate by KP_auth_sign_rsa

- `dir-key-crosscert`
- *CrossSignature*, Object `ID SIGNATURE` or `SIGNATURE`
- Exactly once.
- No extra arguments.

CrossSignature is a signature, made using the certificate's signing key `KP_auth_sign_rsa`, of the PKCS1-padded hash of the certificate's identity key: `SHA1(DER(KP_auth_id_rsa))`. 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` — Signature

- `dir-key-certification`
- RSA signature of the document by KP_auth_id_rsa.
- At end, exactly once.
- No extra argument.

"HSIntro"

The "HSIntro" protocol handles introduction points.

- "3" (`HSINTRO_V2`) – supports the RSA-based introduction point protocol of proposal 121 in Tor 0.2.1.6-alpha.
- "4" (`HSINTRO_V3`) – support ed25519-based HS v3 introduction point protocol as defined by proposal 224 in Tor 0.3.0.4-alpha.
- "5" (`HSINTRO_RATELIM`) – support ESTABLISH_INTRO message DoS parameters extension for onion service version 3 only in Tor 0.4.2.1-alpha.

"HSRend"

The "HSRend" protocol handles rendezvous points.

- "1" (`HSREND_V2`) – supports all features in Tor 0.0.6.
- "2" (`HSREND_V3`) – supports RENDEZVOUS2 messages of arbitrary length as long as they have 20 bytes of cookie in Tor 0.2.9.1-alpha.

"HSDir"

The "HSDir" protocols are the set of hidden service document types that can be uploaded to, understood by, and downloaded from a tor relay, and the set of URLs available to fetch them.

- "1" (`HSDIR_V2`) – supports all features in Tor 0.2.0.10-alpha.
- "2" (`HSDIR_V3`) – support ed25519 blinded keys request which is defined by the HS v3 protocol as part of proposal 224 in Tor 0.3.0.4-alpha.

"DirCache"

The "DirCache" protocols are the set of documents available for download from a directory cache via BEGIN_DIR, and the set of URLs available to fetch them. (This excludes URLs for hidden service objects.)

- "1" (`DIRCACHE_BASE`) – supports all features in Tor 0.2.4.19.

- "2" (`DIRCACHE_CONSDIFF`) – adds support for consensus diffs in Tor 0.3.1.1-alpha.

"Desc"

Describes features present or absent in descriptors.

Most features in descriptors don't require a "Desc" update – only those that need to someday be required. For example, someday clients will need to understand ed25519 identities.

- "1" (`DESC_BASE`) – supports all features in Tor 0.2.4.19.
- "2" (`DESC_CROSSSIGN`) – cross-signing with onion-keys, signing with ed25519 identities.
- "3" (`DESC_NO_TAP`) – parsing relay descriptors without onion-keys; generating them when the `publish-dummy-tap-key` option is `0`.
- "4" – Support for understanding family certs, family IDs, and building paths accordingly.

"Microdesc"

Describes features present or absent in microdescriptors.

Most features in descriptors don't require a "Microdesc" update – only those that need to someday be required. These correspond more or less with consensus methods.

- "1" (`MICRODESC_BASE`) – consensus methods 9 through 20.
- "2" (`MICRODESC_ED25519_KEY`) – consensus method 21 (adds ed25519 keys to microdescs).
- "3" (`MICRODESC_NO_TAP`) – Accepts Microdescriptors without onion-key bodies. (Consensus method TBD; see proposal 350.)

"Cons"

Describes features present or absent in consensus documents.

- Exactly once.

date and ***time*** are as for `published` in a router descriptor.

The time when this document and corresponding key were last generated.

Implementations SHOULD reject certificates that are published too far in the future, though they MAY tolerate some clock skew.

`dir-key-expires` – Certificate expiry time

- `dir-key-expires` *date time* ..
- Exactly once.

The time after which this certificate is no longer valid. ***date*** and ***time*** are as for `dir-key-published`.

Implementations SHOULD reject expired certificates, though they MAY tolerate some clock skew.

`dir-identity-key` — authority identity key, KP_auth_id_rsa

- `dir-identity-key`
- *key*_ Object, `RSA PUBLIC KEY`
- Exactly once.
- No extra arguments

The long-term authority identity key KP_auth_id_rsa for this authority. ***key*** is a DER PKCS#1 RSAPublicKey structure encoded as an Object.

This key SHOULD be at least 2048 bits long; it MUST NOT be shorter than 1024 bits.

`dir-signing-key` — Signing key, KP_auth_sign_rsa

- `dir-signing-key`
- *key*_ Object, `RSA PUBLIC KEY`
- Exactly once
- No extra arguments.

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

Directory authority key certificates

Directory authorities create key certificates to certify their medium-term signing keys (`KP_auth_sign_rsa`) with their long-term authority identity keys (`KP_auth_id_rsa`).

An authority key certificate is a netdoc. Authority key certificates can appear as a sub-section of other documents, notably network status votes.

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

Authority key certificate items

`dir-key-certificate-version` — Introduce an auth key cert

- `dir-key-certificate-version` *version* ..
- At start, exactly once

States the protocol version of the key certificate.

`version` MUST be `3`. Implementations MUST reject formats they don't understand.

`dir-address` — Public directory service address

- `dir-address` *address*:*port* ..
- At most once

The IP address and TCP port at which this authority serves directory requests over HTTP,

`fingerprint` — authority identity, H(KP_auth_id_rsa)

- `fingerprint` *fingerprint* ..
- Exactly once.

fingerprint is SHA1(DER(KP_auth_id_rsa)), in uppercase hex.

`dir-key-published` — Certificate generation time

- `dir-key-published` *date time* ..

Most features in consensus documents don't require a "Cons" update – only those that need to someday be required.

These correspond more or less with consensus methods.

- "1" (`CONS_BASE`) – consensus methods 9 through 20.
- "2" (`CONST_ED25519_MDS`) – consensus method 21 (adds ed25519 keys to microdescs).

"Padding"

Describes the padding capabilities of the relay.

- "1" [DEFUNCT] – Relay supports circuit-level padding. This subprotocol MUST NOT be used as it was also enabled in relays that don't actually support circuit-level padding. Advertised by Tor versions from tor-0.4.0.1-alpha and only up to and including tor-0.4.1.4-rc.
- "2" (`PADDING_MACHINES_CIRC_SETUP`) – Relay supports the HS circuit setup padding machines (proposal 302). Advertised by Tor versions from tor-0.4.1.5 and onwards.

"FlowCtrl"

Describes the flow control protocol at the circuit and stream level. If there is no FlowCtrl advertised, tor supports the unauthenticated flow control features (version 0).

- "1" (`FLOWCTRL_AUTH_SENDME`) – supports authenticated circuit level SENDMEs as of proposal 289 in Tor 0.4.1.1-alpha.
- "2" (`FLOWCTRL_CC`) – supports congestion control by the Exits which implies a new SENDME format and algorithm. See proposal 324 for more details. Advertised in tor 0.4.7.3-alpha.

"Conflux"

Describes the communications mechanisms used to bundle circuits together, in order to split traffic across multiple paths.

- "1" (`CONFLUX_BASE`) – Supports the base coflux protocol from proposal 329.

Directory authority operation and formats

Every authority has two keys used in this protocol:

`KP_auth_id_rsa`: a long-term RSA authority identity key. This key SHOULD be at least 2048 bits long; it MUST NOT be shorter than 1024 bits.

`KP_auth_sign_rsa`: a medium-term RSA authority signing key. This key SHOULD be at least 2048 bits long; it MUST NOT be shorter than 1024 bits.

The identity key `KP_auth_id_rsa` is used from time to time to sign new key certificates containing (and authenticating) new `KP_auth_sign_rsa` signing keys; it is very sensitive. It may be kept offline.

The signing key `KP_auth_sign_rsa` is used to sign consensuses, votes, and similar documents.

(Authorities also have a router identity key `KP_relayid_rsa`, and other keys used in their role as a router, and by earlier versions of the directory protocol.)

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]) encoding a relay's `SHA1(DER(KP_relayid_rsa))`.

bool ::= "0" | "1"

Binary certificate formats

This document describes two certificate formats that Tor uses for certifying Ed25519 keys, and discusses how those formats is labeled and encoded.

There are:

- "Tor Ed25519 certificates", which are signed by Ed25519 keys.
- "Tor RSA→Ed25519 cross-certificates", which are signed by RSA keys.

These are not the only certificate format that Tor uses. For the certificates that authorities use for their signing keys, see "Authority key certificates".

Additionally, Tor uses TLS, which depends on X.509 certificates.

> The certificates in this document were first introduced in proposal 220, and were first supported by Tor in Tor version 0.2.7.2-alpha.

Tor Ed25519 Certificates

A Tor Ed25519 Certificate certifies a key, or a digest of a key, or a digest of some other object, using an Ed25519 key to sign it.

Field	Size	Description
`VERSION`	1	The version of this format
`CERT_TYPE`	1	Purpose and meaning of the cert
`EXPIRATION_DATE`	4	When the cert becomes invalid
`CERT_KEY_TYPE`	1	Type of `CERTIFIED_KEY`
`CERTIFIED_KEY`	32	Certified key, or its digest
`N_EXTENSIONS`	1	Number of extensions
`N_EXTENSIONS` times:		
- `ExtLen`	2	Length of encoded extension body
- `ExtType`	1	Type of extension
- `ExtFlags`	1	Control interpretation of extension
- `ExtData`	`ExtLen`	Encoded extension body
`SIGNATURE`	64	Signature of all previous fields

The `VERSION` field holds the value `[01]`.

The CERT_TYPE field holds a value depending on the type of certificate. (See "Certificate types".)

The CERTIFIED_KEY field is a subject public key, or a digest of a subject public key. The representation depends on the value of CERT_KEY_TYPE. (See "List of certified key types".)

> Note that the Tor Ed25519 Certificate format can only certify objects of length 32. To sign a longer key, we compute its digest. But if we need to make certificates that includes the full value of a longer key, we'll have to use a new extension, or a new certificate.

The EXPIRATION_DATE is a date, given in **hours** since the epoch, after which this certificate isn't valid.

> (A four-byte date here will work fine until 10136 A.D.)

The ExtFlags field holds flags. Only one flag is currently defined:

- **1**: AFFECTS_VALIDATION. This flag tells an implementation how to handle an extension whose ExtType it does not recognize. If this flag is set on an extension, then the extension affects whether the certificate is valid; implementations MUST NOT accept the certificate as valid if the do not recognize the extension's ExtType. If this flag is *not* set on an extension, implementations MUST ignore that extension if they do not recognize the ExtType.

> Ignoring a *recognized* extension is never valid.

The interpretation of ExtBody depends on the ExtType field. See "Recognized extensions" below.

It is an error for an ExtLen to extend beyond the end of a certificate.

Before acting based on any certificate, parties MUST know which key it is supposed to be signed by, and then check the signature.

> It *is* okay to inspect the certificate's fields before checking the signature, and reject the certificate if is ill-formed, expired, or so on.
>
> This does not count as "acting based on a certificate."

The "{read|write}-overload-count" are the counts of how many times the reported limits of burst/rate were exhausted and thus the maximum between the read and write count occurrences. To make the counter more meaningful and to avoid multiple connections saturating the counter when a relay is overloaded, we only increment it once a minute.

The 'version' field is set to '1' for now.

(Introduced in tor-0.4.6.1-alpha)

```
"overload-fd-exhausted" SP version YYYY-MM-DD HH:MM:SS NL
    [At most once.]
```

Indicates that a file descriptor exhaustion was experienced by this relay.

The timestamp indicates that the maximum was reached between the timestamp and the "published" timestamp of the document.

This overload field should remain in place for 72 hours since last triggered. If the limits are reached again in this period, the timestamp is updated, and this 72 hour period restarts.

The 'version' field is set to '1' for the initial implementation which detects fd exhaustion only when a socket open fails.

(Introduced in tor-0.4.6.1-alpha)

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

"router-signature" NL Signature NL
    [At end, exactly once.]
    [No extra arguments]
```

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.

YYYY-MM-DD HH:MM:SS defines the end of the included measurement interval of length NSEC seconds (86400 seconds by default). Counts are reset to 0 at the end of this interval.

The keyword list is currently as follows:

```
          bin-size
            - The current rounding value for cell count fields (10000
by
              default)
          write-drop
            - The number of RELAY_DROP messages this relay sent
          write-pad
            - The number of PADDING cells this relay sent
          write-total
            - The total number of cells this relay cent
          read-drop
            - The number of RELAY_DROP messages this relay received
          read-pad
            - The number of PADDING cells this relay received
          read-total
            - The total number of cells this relay received
          enabled-read-pad
            - The number of PADDING cells this relay received on
              connections that support padding
          enabled-read-total
            - The total number of cells this relay received on
connections
              that support padding
          enabled-write-pad
            - The total number of cells this relay received on
connections
              that support padding
          enabled-write-total
            - The total number of cells sent by this relay on
connections
              that support padding
          max-chanpad-timers
            - The maximum number of timers that this relay scheduled
for
              padding in the previous NSEC interval

    "overload-ratelimits" SP version SP YYYY-MM-DD SP HH:MM:SS
                        SP rate-limit SP burst-limit
                        SP read-overload-count SP write-overload-count
NL
        [At most once.]

        Indicates that a bandwidth limit was exhausted for this relay.
```

The “rate-limit” and “burst-limit” are the raw values from the BandwidthRate and BandwidthBurst found in the torrc configuration file.

The signature is created as an Ed25519 signature, over all the fields in the certificate up until but not including `SIGNATURE`.

> Note that this signature is not personalized with a prefix string. That’s probably an oversight that we should correct if we make another cert format in the future.

Defined Tor Ed25519 certificate extensions

Signed-with-ed25519-key extension [type 04]

In several places, it’s desirable to bundle the signing key along with the certificate. We do so with this extension.

With this extension:

- `ExtLen` is 32.
- `ExtData is a 32-byte Ed25519 public key.

When this extension is present, it MUST match the key used to sign the certificate.

Tor RSA→Ed25519 cross-certificate

A Tor RSA→Ed25519 Cross-certificate signs an Ed25519 key using a legacy 1024-bit RSA key. Its format is:

Field	Size	Description
`ED25519_KEY`	32	The subject key
`EXPIRATION_DATE`	4	When the cert becomes invalid
`SIGLEN`	1	Length of RSA signature.
`SIGNATURE`	`SIGLEN`	RSA Signature

Just as with the Ed25519 certificates above, the `EXPIRATION_DATE` field is a number of **hours** since the epoch.

As elsewhere, the RSA signature is generated using RSA-PKCSv1 padding, with hash algorithm OIDs omitted.

The signature is computed on the SHA-256 digest of `PREFIX | FIELDS`, where `PREFIX` is the string `"Tor TLS RSA/Ed25519 cross-certificate"` (without any

terminating NUL), and FIELDS is all other fields in the certificate (other than the signature itself).

Certificate types (CERT_TYPE field)

This table shows values that can appear in the CertType field used in a CERTS cell during channel negotiation.

Some of these values (those marked with "Ed") are also used in the CERT_TYPE field in Tor Ed25519 certificates. (X.509 and RSA→Ed25519 cross-certificates don't have a CERT_TYPE field.)

> You might need to scroll this table to view it all.
>
> We'll try to fix this once we have a better grip on our mdbook CSS.

Type	Mnemonic	Format	Subject	Signing k
[01]	TLS_LINK_X509	X.509	KP_legacy_conn_tls	KS_relay
[02]	RSA_ID_X509	X.509	KP_relayid_rsa	KS_relay
[03]	LINK_AUTH_X509	X.509	KP_legacy_linkauth_rsa	KS_relay
[04]	IDENTITY_V_SIGNING	Ed	KP_relaysign_ed	KS_relay
[05]	SIGNING_V_TLS_CERT	Ed	A TLS certificate	KS_relays
[06]	SIGNING_V_LINK_AUTH	Ed	KP_link_ed	KS_relays
[07]	RSA_ID_V_IDENTITY	Rsa	KP_relayid_ed	KS_relay
[08]	BLINDED_ID_V_SIGNING	Ed	KP_hs_desc_sign	KS_hs_bl
[09]	HS_IP_V_SIGNING	Ed	KP_hs_ipt_sid	KS_hs_des
[0A]	NTOR_CC_IDENTITY	Ed	KP_relayid_ed	EdCvt(KS_
[0B]	HS_IP_CC_SIGNING	Ed	KP_hss_ntor	KS_hs_des

smaller integer and included as 'NUM'. Note that the overall reported value can be negative.

(Introduced in tor-0.4.6.1-alpha)

```
"hidserv-dir-onions-seen" SP NUM SP key=val SP key=val ... NL
    [At most once.]
"hidserv-dir-v3-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.

(Introduced in tor-0.4.6.1-alpha)

```
"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.

```
    "transport-info" [version=VersionString]
[implementation=ImplementationString] NL
        [Any number.]
```

There will be one "transport-info" line after one or multiple "tranport" lines descriving the implmentation of those transport lines. It can optionally contain the version in the 'VersionString' and the implementation in the 'ImplementationString' or be empty if the pluggable transport doesn't inform of it's implementation.

```
    "padding-counts" YYYY-MM-DD HH:MM:SS (NSEC s) key=NUM key=NUM ...
NL
        [At most once.]
```

```
"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.]
"hidserv-v3-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.

(Introduced in tor-0.4.6.1-alpha)

```
"hidserv-rend-relayed-cells" SP NUM SP key=val SP key=val ... NL
    [At most once.]
"hidserv-rend-v3-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

Type	Mnemonic	Format	Subject	Signing k
[[0C]	FAMILY_V_IDENTITY	Ed	KP_relayid_ed	KS_family

Note 1: The certificate types `[09] HS_IP_V_SIGNING` and `[0B] HS_IP_CC_SIGNING` were implemented incorrectly, and now cannot be changed. Their signing keys and subject keys, as implemented, are given in the table. They were originally meant to be the inverse of this order.

List of extension types

- `[04]` - signed-with-ed25519-key

List of certified key types (CERT_KEY_TYPE field)

- `[01]`: ed25519 key
- `[02]`: `SHA256(DER(key))` for an RSA key. (Not currently used.)
- `[03]`: SHA-256 digest of an X.509 certificate. (Used with certificate type 5.)

(NOTE: Up till 0.4.5.1-alpha, all versions of Tor have incorrectly used `[01]` for all types of certified key. Implementations SHOULD allow “01” in this position, and infer the actual key type from the `CERT_TYPE` field.

Tor directory protocol, version 3

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.

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

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.

This measurement includes both IPv4 and IPv6 connections.

```
    "ipv6-conn-bi-direct" YYYY-MM-DD HH:MM:SS (NSEC s)
BELOW,READ,WRITE,BOTH NL
        [At most once]
```

Number of IPv6 connections that are used uni-directionally or bi-directionally. See “conn-bi-direct” for more details.

```
    "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).

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=NUM,CC=NUM,... 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)

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.

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.
```

```
        "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).

```
    "dirreq-v2-resp" status=NUM,... NL
        [At most once.]
    "dirreq-v3-resp" status=NUM,... 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=NUM,... NL
        [At most once.]
    "dirreq-v3-direct-dl" key=NUM,... NL
        [At most once.]
    "dirreq-v2-tunneled-dl" key=NUM,... NL
        [At most once.]
    "dirreq-v3-tunneled-dl" key=NUM,... 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 relay message 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:

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.

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.

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. Each authority uploads their vote to all other authorities.

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

`DistSeconds` and `VoteSeconds` are carried in the `voting-delay` item in the consensus.

Authorities SHOULD also reject any votes that other authorities try to upload after this time. (0.4.4.1-alpha was the first version to reject votes in this way.)

Note: Refusing late uploaded votes minimizes the chance of a consensus split, particular when authorities are under bandwidth pressure. If an authority is struggling to upload its vote, and finally uploads to a fraction of authorities after this period, they will compute a consensus different from the others. By refusing uploaded votes after this time, we increase the likelihood that most authorities will use the same vote set.

(extra-info document items in ad-hoc representation)

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

`bridge-ips` — Bridge country code stats

- `bridge-ips` *CC=NUM*,*CC=NUM*,... ..
- First argument is one or more *CC=NUM*, comma-separated.
- At most once

List of mappings from two-letter country codes ***CC*** to the number of unique IP addresses ***NUM*** 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` — Bridge IP protocol version stats

- `bridge-ip-versions` *VER=NUM*,*VER=NUM*,... ..
- First argument is one or more *VER=NUM*, comma-separated.
- At most once

List of counts ***NUM*** of unique IP addresses that have connected to the bridge per IP protocol version ***VER*** (`4` or `6`).

`bridge-ip-transports` — Bridge pluggable transport stats

- `bridge-ip-transports` *PT=NUM*,*PT=NUM*,.....
- First argument is zero or more *PT=NUM*, comma-separated.
- 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.

Rejecting late uploaded votes does not fix the problem entirely. If some authorities are able to download a specific vote, but others fail to do so, then there may still be a consensus split. However, this change does remove one common cause of consensus splits.

VA-DistSeconds: The authorities calculate the consensus and exchange signatures. (This is the earliest point at which anybody can possibly get a given consensus if they ask for it.)

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; clients should continue to try to download a new consensus if they have not done so already.

VU + 24 hours: Clients will no longer use the consensus at all.

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

netdoc document meta-format

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

netdoc syntax

A netdoc is a text file, with Unix line endings.

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-], but may not start with -. An Object is a block of encoded data in pseudo-Privacy-Enhanced-Mail (PEM) style format: that is, lines of encoded data MAY be wrapped by inserting an ascii linefeed ("LF", also called newline, or "NL" here) character (cf. RFC 4648 §3.1). When line wrapping, implementations MUST wrap lines at 64 characters. Upon decoding, implementations MUST ignore and discard all linefeed characters.

A netdoc consists of unicode code points, and MUST be encoded as UTF-8 without a BOM prefix. Implementations SHOULD reject netdocs that are not UTF-8, or which contain the NUL character.

> **Future directions**:
>
> (Note that we may impose additional restrictions on the set of allowable Unicode characters in future, to restrict control characters and other oddities.)

> **Conformance**:
>
> Arti currently rejects all non-UTF-8 documents.
>
> C Tor directory authorities (as of 0.4.8.x) reject non-UTF-8 and UTF-with-BOMs when receiving router descriptors; C tor accepts arbitrary non-NUL byte sequences otherwise.

More formally:

Declare how much bandwidth the OR has used recently, on IPv6 connections. See "read-history" and "write-history" for full details.

`geoip-db-digest`, `geoip6-db-digest` — Digests of GeoIP databases

- `geoip-db-digest` *sha1-digest* ..
- `geoip6-db-digest` *sha1-digest* ..
- At most once each

sha1-digest is the SHA1 digest of the GeoIP database file that is used to resolve IPv4 or IPv6 addresses (respectively) to country codes, encoded in hexadecimal.

`geoip-start-time`, `geoip-client-origins` — Obsolete usage info

("geoip-start-time" YYYY-MM-DD HH:MM:SS NL) ("geoip-client-origins" CC=NUM,CC=NUM,... 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-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` — Bridge stats interval

- `bridge-stats-end` *YYYY-MM-DD HH:MM:SS* (*NSEC* s)
- At most once
- Anomalous argument format

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` item, as well as any other `bridge-*` item, is only added when the relay has been running as a bridge for at least 24 hours.

Extra-info document format

A server descriptor can reference an extra-info netdoc, by specifying an `extra-info-digest` Item.

extra-info items

Some Items are precisely as in the router descriptor:

- `identity-ed25519`
- `published`

`extra-info` — Introduce a server's extra-info

- `extra-info` *Nickname Fingerprint* ..
- 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.

`read-history`, `write-history`, `ipv6-read-history`, `ipv6-write-history` — Recent bandwidth use

```
    "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.

These fields include both IPv4 and IPv6 traffic.

```
    "ipv6-read-history" YYYY-MM-DD HH:MM:SS (NSEC s) NUM,NUM,NUM... NL
        [At most once]
    "ipv6-write-history" YYYY-MM-DD HH:MM:SS (NSEC s) NUM,NUM,NUM...
NL
        [At most once]
```

```
NL = The ascii LF character (hex value 0x0a).
Document ::= (Item | NL)+
Item ::= KeywordLine Object?
KeywordLine ::= (Opt WS)? ItemKeyword (WS Arguments)? NL
ItemKeyword = Keyword
Arguments ::= Any sequence of unicode characters encoded in UTF-8,
excluding NL and NUL.
WS = (SP | TAB)+
Object ::= BeginLine Base64-encoded-data EndLine
BeginLine ::= "-----BEGIN " Keyword (" " Keyword)*"-----" NL
EndLine ::= "-----END " Keyword (" " Keyword)* "-----" NL
Keyword = KeywordStart KeywordChar*
KeywordStart ::= 'A' ... 'Z' | 'a' ... 'z' | '0' ... '9'
KeywordChar ::= KeywordStart | '-'
Opt ::= "opt"
```

The documentation for each ItemKeyword must specify its expected Arguments and Objects. Unless otherwise stated, a KeywordLine contains a sequence of space/tab-separated arguments:

```
Arguments ::= Argument (WS Arguments)?
Argument := ArgumentChar+
ArgumentChar ::=  Any unicode characters encoded in UTF-8,
          excluding NL, NUL, TAB, and SP.
```

A ItemKeyword may not be `opt`.

Implementations MUST NOT generate "Opt" in a keyword line, though they SHOULD accept it.

> **Conformance:**
>
> Some implementations do not accept Opt on all items. Notably, C Tor will reject many netdocs if they use "Opt" on an KeywordLine used to indicate the start or end of a section, or an a KeywordLine containing a signature.

> Before Tor 0.1.2.5-alpha, Opt was used to indicate that if a parser did not recognize an ItemKeyword, it should ignore it. Now all unrecognized ItemKeywords are treated that way.
>
> In Tor 0.2.0.5-alpha through 0.2.4.1-alpha we stopped generating Opt. No currently supported Tor release generates it.

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

Compatibility and extensibility

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.

Permit additional arguments

For forward compatibility, each item MUST allow extra arguments at the end of the line unless otherwise noted. So, for example, if an item's description is given as:

- `thing` **int int int ..**

then implementations SHOULD accept this string as well:

`thing 5 9 11 13 16 12` NL

but not this string:

`thing 5` NL

Typically the text would state that the `int` arguments are integers, so the implementation should also reject this string:

`thing 5 10 thing` NL

Whenever an item DOES NOT allow extra arguments, we will tag it with **"No extra arguments"** in the syntax bullet points. (If the .. has been omitted, but there is no "no extra arguments" statement, the omission of the .. is a spec mistake and extra arguments *are* allowed.)

netdoc structure

Each type of netdoc requires, and permits, certain ItemKeywords, with certain restrictions on their order. In some cases ItemKeywords can introduce sections, providing structure to the document; this will be stated in the description for that ItemKeyword in that type of document.

This digest is computed over:

- the fixed string `Tor router descriptor signature v1`;
- the contents of the document, starting at the beginning,
- but, only up to and including the first space after the `router-sig-ed25519` keyword.

Note that this differs in several respects from the definition of a signature item.

The signature is encoded in Base64, with terminating =s removed.

[Before Tor 0.4.5.1-alpha, this field was optional whenever identity-ed25519 was absent.]

`router-signature` — RSA signature

- `router-signature`
- *SIGNATURE* Object, `SIGNATURE`
- At end, exactly once
- No extra arguments
- RSA signature of the document by KP_relayid_rsa

The digest includes the `router-sig-ed25519` item.

```
       Present if the router accepts "tunneled" directory requests
using a
       BEGIN_DIR relay message over the router's OR port.
          (Added in 0.2.8.1-alpha. Before this, Tor relays accepted
          tunneled directory requests only if they had a DirPort open,
          or if they were bridges.)
```

`proto` - Subprotocol capabilities supported

- `proto` *entry entry* ..
- Exactly once.
- Zero or more *entry* arguments.

Syntax of each *entry*:

```
entry = Keyword "=" Values

Values =
Values = Value
Values = Value "," Values

Value = Int
Value = Int "-" Int

Int = NON_ZERO_DIGIT
Int = Int DIGIT
```

Each ***entry*** indicates that the Tor relay supports all of the subprotocols capabilities in question. (See Subprotocol-based versioning for a definition of recognized keywords and values.) Entries should be sorted by keyword. Values should be numerically ascending within each entry. (This implies that there should be no overlapping ranges.) Ranges should be represented as compactly as possible. _Int_s must be no larger than 63.

This field was first added in Tor 0.2.9.x.

[Before Tor 0.4.5.1-alpha, this field was optional.]

`router-sig-ed25519` — Signature

- `router-sig-ed25519` *Signature*
- Exactly once, at end, just before `router-signature`

Ed25519 signature by K_relaysign_ed (as certified in `identity-ed25519`) on the SHA256 digest of the document, as follows:

netdoc format description conventions

NB these conventions are not yet followed everywhere in the Tor Specifications.

When presenting a specific document format, the Items forming the document are shown one per subsection.

The syntax of each item is defined in detail with a bulleted list at the start of the section.

The first bullet point shows the syntax of the line introducing the item. Literal parts (including the Item Keyword) are shown in `fixed width`. Arguments are shown with *italic emphasis*. The spaces between arguments, and the final newline, are not depicted. If (as is usual) extra arguments are to be tolerated (for future expansion), a short ellipsis .. is shown as a reminder. Optional arguments are shown in [ ].

When an Item has (or may have) an Object, that is shown as the 2nd line in the bullet list, in the form:

- *something*, Object, `OBJECT KEYWORD` where *something* will be used to refer to the Object in the text, and `OBJECT KEYWORD` is the Object's Keyword in the base64 delimiters. (The `-----BEGIN` etc. are not depicted.)

Further bullet points give further information about the syntax - often, in terms defined more fully here.

The type (therefore, format) of arguments, and permissible values, are stated in the text. The argument is named in ***bold-italic*** in its principal description.

Position and multiplicity

The syntax bullet points for an Item state its permissible multiplicity and position, within each Document of its particular document type, in the following terms:

- **"At start, exactly once"** — MUST occur exactly once, and MUST be the first item.
- **"Exactly once"** — MUST occur exactly once.
- **"At end, exactly once"** — MUST occur exactly once, and MUST be the last item.
- **"At most once"** — MAY occur zero or one times but MUST NOT occur more than once.

- **“Any number”** — MAY occur zero, one, or more times.
- **“Once or more”** — MUST occur at least once and MAY occur more than once.

Rest-of-line arguments

Exceptionally, for some items there is a “rest of line” argument. This is denoted by writing ARGUMENT.... in the syntax summary, in the first bullet point, and stating

- *ARGUMENT* is the whole rest of the line,

in the syntax description.

In this case, the value of the argument is all the characters after the SP following the keyword or previous argument.

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 PKCS#1 1.5 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.

The signature does not include the algorithmIdentifier specified in PKCS #1.

Unless specified 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*.

If a document may contain multiple signatures, it will be explicitly stated which signature item(s) are included in the digest(s) of which other signature item(s).

`allow-single-hop-exits` — Declare support for single-hop exit circuits

“allow-single-hop-exits” NL

[At most once.] [No extra arguments]

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. This is obsolete in tor version >= 0.3.1.0-alpha.

`or-address` – Alternative ORport address and port

“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 address/port pairs. As of Tor 0.2.3.x only the first address/port pair is advertised and used.

`tunnelled-dir-server` — Accepts BEGIN_DIR relay message via ORport

“tunnelled-dir-server” NL

[At most once.] [No extra arguments]

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

   "caches-extra-info" NL
```

[At most once.] [No extra arguments]

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` — Hash of the extra-info document

- `extra-info-digest` *sha1-digest* [*sha256-digest*] ..
- At most once

sha1-digest is the SHA1 digest, hex-encoded 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 Item is absent, the router does not have an extra-info document.

sha256-digest is the SHA256 digest, base64-encoded, of the extra-info document. Unlike the *sha1-digest*, this digest is calculated over the entire document, including the signature.

> The difference in the inputs to the two digests is due to a long-lived bug in the tor implementation that it would be difficult to roll out an incremental fix for, not a design choice. Future digest algorithms specified should not include the signature in the data used to compute the digest.

[Versions before 0.2.7.2-alpha did not include a SHA256 digest.] [Versions before 0.2.0.1-alpha don't recognize this field at all.]

`hidden-service-dir` — Declares this router to be a Hidden Service directory

- `hidden-service-dir` ..
- At most once

Present only if this router stores and serves hidden service descriptors. This router supports the descriptor versions declared in the HSDir "proto" entry. If there is no "proto" entry, this router supports version 2 descriptors.

Router operation and formats

This section describes how relays must behave when publishing their information to the directory authorities, and the formats that they use to do so.

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.

      - Its uptime is less than 24h and bandwidth has changed by a
factor of 2
        from the last time a descriptor was generated, and at least a
given
        interval of time (3 hours by default) has passed since then.

      - Its uptime has been reset (by restarting).

      - It receives a networkstatus consensus in which it is not
listed.

      - It receives a networkstatus consensus in which it is listed
        with the StaleDesc flag.

      [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).

Tor versions older than 0.3.5.1-alpha ignore uptime when checking for bandwidth changes.

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.

Each ***name*** is a relay nickname, or hexdigest. 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.

Relays should omit this entry if the `publish-family-list` parameter is 0.

`family-cert` — Prove membership in a relay family

- `family-cert`
- *cert* Object, `FAMILY CERT`.
- Any number of times

The *cert* object is a *family certificate*, an ed25519 certificate proving this relay's membership in the family corresponding to the certificate's signing key.

In addition to regular validity and liveness constraints, the certificate must have these properties:

- It must have its `CERT_TYPE` field set to `FAMILY_V_IDENTITY`.
- It must include its signing key (KP_familyid_ed) in a signed-with-ed25519-key extension.
- Its certified key must be the same as the relay's KP_relayid_ed key, as listed in the identity-ed25519 entry.

A certificate meeting these constraints proves that the relay is a member of a family with the family ID `ed25519:`**FID**, where `FID` is the unpadded base-64 encoding of the certificate's KP_relayid_ed key.

`eventdns` — Declare support for non-obsolete DNS logic

"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.

INFO is starts with the first non-whitespace after the whitespace after `contact`, and is the whole rest of the line up to but not including the newline.

`bridge-distribution-request` — Request distribution method

- `bridge-distribution-request` *Method* ..
- At most once
- Bridges only

Method describes how a Bridge address is distributed. Recognized methods are:

- `none` — The distributor will avoid distributing your bridge address;
- **`any``** — The distributor will choose how to distribute your bridge address;
- `https` — specifies distribution via the web interface at https://bridges.torproject.org;
- `email` — specifies distribution via the email autoresponder at bridges@torproject.org;
- `moat` — specifies distribution via an interactive menu inside Tor Browser.

Potential future *Method*s must be as follows:

```
Method = (KeywordChar | "_") +
```

All bridges SHOULD include this line. Non-bridges MUST NOT include it.

The bridge distributor SHOULD treat unrecognized Method values as if they were “none”.

(Default: “any”)

[This line was introduced in 0.3.2.3-alpha, with a minimal backport to 0.2.5.16, 0.2.8.17, 0.2.9.14, 0.3.0.13, 0.3.1.9, and later.]

> As of 2025, the bridge database is rdsys, which chooses from among many distribution methods. Previously, we used BridgeDB.

`family` — Group relays for the purposes of path selectdion

- `family` *name name* ..
- One or more *name* arguments
- At most once

Server descriptor format

Server descriptors consist of the following items.

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.

Note that many versions of Tor will generate an extra newline at the end of their descriptors. Implementations MUST tolerate one or more blank lines at the end of a single descriptor or a list of concatenated descriptors. New implementations SHOULD NOT generate such blank lines.

Server descriptor items

`router` — Introduce a router descriptor

- `router` ***nickname address ORPort SOCKSPort DirPort ..***
- 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` — Specify the router’s ed25519 identity

- `identity-ed25519`
- *certificate* Object, `ED25519 CERT`
- Exactly once, in second position in document.
- No extra arguments.

certificate is an Ed25519 certificate on KP_relaysign_ed by KP_relayid_ed with terminating =s removed from its base64-encoding.

When this element is present, it MUST appear as the first or second element in the router descriptor.

certificate has CERT_TYPE of [04]. It must include a signed-with-ed25519-key extension so that we can extract the master identity key.

[Before Tor 0.4.5.1-alpha, this field was optional.]

`master-key-ed25519` — Redundantly specify the router's ed25519 identity

- `master-key-ed25519` ***MasterKey*** **..**
- Exactly once.

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

[Before Tor 0.4.5.1-alpha, this field was optional.]

`bandwidth` — Report router's network bandwidth

- `bandwidth` ***bandwidth-avg bandwidth-burst bandwidth-observed*** **..**
- Exactly once.
- Each argument is a number in decimal, in bytes per second.

Estimated bandwidth for this router. ***bandwidth-avg*** is the volume that the OR is willing to sustain over long periods; ***bandwidth-burst*** is the volume that the OR is willing to sustain in very short intervals.

bandwidth-observed 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 5 days, and another sustained input. *bandwidth-observed* value is the lesser of these two numbers.

Tor versions released before 2018 only kept bandwidth-observed for one day. These versions are no longer supported or recommended.

`platform` — Describe the platform on which this relay is running

- `platform` *string...*
- *string* is the whole rest of the line
- At most once
- At most once

An exit-policy summary summarizing the router's rules for connecting to IPv6 addresses. A missing `ipv6-policy` line is equivalent to `ipv6-policy reject 1-65535`.

`overload-general` — Relay is overloaded

- `overload-general` *version YYYY-MM-DD HH:MM:SS*
- At most once

Indicates that a relay has reached an "overloaded state" which can be one or many of the following load metrics:

- Any OOM invocation due to memory pressure
- Any ntor onionskins are dropped
- TCP port exhaustion

The timestamp is when overload was detected by at least one metric. It should always be on the hour; so for example `2020-01-10 13:00:00` is an expected timestamp. Because this is a binary state, if the line is present, we consider that it was hit at the very least once somewhere between the `overload-general` timestamp, and the server descriptor's `published` timestamp which is when the document was generated.

The `overload-general` line should remain in place for 72 hours after last triggered. If the limits are reached again in this period, the timestamp is updated, and this 72 hour period restarts.

The `version` field is set to `1` for now.

(Introduced in tor-0.4.6.1-alpha, but moved from extra-info to general descriptor in tor-0.4.6.2-alpha)

`contact` — Server administrator contact information

- `contact` INFO....
- *INFO* is the whole rest of the line
- At most once

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

The encoding is as for “onion-key” above.

`accept`, `reject` — Exit policy

- `accept` exitpattern
- `reject`“ exitpattern
- Any number

These lines describe the “exit policy”: the rules that this OR follows when deciding whether to allow a new stream to a given address.

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 *:*.

The syntax is as follows:

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

`ipv6-policy` — Exit policy summary for IPv6

- `ipv6-policy` `accept`|`reject` *PortList*

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` — Time this descriptor (and extra-info) was generated

- `published` *date time* ..
- *date time* is *YYYY-MM-DD HH*:*MM*:*SS* and is in UTC
- Exactly once

When this descriptor (and its corresponding extra-info document if any) was generated.

`fingerprint` – Redundant hash of ASN-1-encoding of router identity key

- `fingerprint` *fingerprint* ..
- *fingerprint* is multiple arguments — see below.
- At most once

fingerprint is the hash `SHA1(DER(KP_relayid_rsa))`, encoded in hex, with a single space after every 4 characters. A descriptor is considered invalid (and MUST be rejected) if the fingerprint line does not match the public key.

`hibernating` — Whether the relay is hibernating

- `hibernating` bool ..
- 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.

`uptime` — How long this relay has been continuously running

- `uptime` number
- At most once

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

`onion-key` — Relay’s obsolete RSA TAP key

- `onion-key`
- *key* Object, `RSA PUBLIC KEY`
- At most once

- No extra arguments

This element MUST be present if `onion-key-crosscert` is present. Relays MUST generate this element unless the `publish-dummy-tap-key` network parameter is set to 0.

This obsolete RSA key was once used used to encrypt CREATE cells. It is no longer used. The key, if present, MUST be 1024 bits. Clients SHOULD validate this element if it is provided.

key is a DER PKCS#1 RSAPublicKey structure encoded as an Object.

`onion-key-crosscert` — Reverse signature by obsolete TAP key

- `onion-key-crosscert`
- *signature* cert NL a RSA signature in PEM format.
- At most once
- No extra arguments

This element MUST be present if onion-key is present. Clients SHOULD validate this element if it is provided.

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

```
A SHA1 hash of the RSA identity key KP_relayid_rsa
   from "signing-key" (see below) [20 bytes]
The Ed25519 identity key KP_relayid_ed
   from "master-key-ed25519" [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.

[Before Tor 0.4.5.1-alpha, this field was optional whenever identity-ed25519 was absent.]

`ntor-onion-key` – KP_ntor, the circuit extension key

- `ntor-onion-key` *KP_ntor* ..
- *KP_ntor* is base64 (and MAY have the =-padding).
- Exactly once

KP_ntor is the curve25519 public key used for the ntor circuit extended handshake.

The key MUST be accepted for at least 1 week after any new key is published in a subsequent descriptor.

[Before Tor 0.4.5.1-alpha, this field was optional.]

`ntor-onion-key-crosscert` — Reverse cert by K_ntor on KP_relayid_ed

- `ntor-onion-key-crosscert` *Bit* ..
- *certificate*, Object `ED25519 CERT`
- Exactly once
- No extra arguments

certificate is an Ed25519 certificate created with the ntor-onion-key, 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. If *Bit* is “0”, then implementations MUST guarantee that the x-coordinate of the resulting ed25519 public key is positive. Otherwise, if *Bit* is “1”, then the sign of the x-coordinate MUST be negative.

To compute the ed25519 public key corresponding to a curve25519 key, and for further explanation on key formats, see appendix C.

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

[Before Tor 0.4.5.1-alpha, this field was optional whenever identity-ed25519 was absent.]

`signing-key` — KP_relayid_rsa, relay’s obsolete RSA identity key

- `signing-key`
- *KP_relayid_rsa*, Object `RSA PUBLIC KEY`
- Exactly once
- No extra arguments

KP_relayid_rsa is the OR’s long-term RSA identity key. It MUST be 1024 bits.