Workgroup: Internet Engineering Task Force
Internet-Draft: draft-mlichvar-ntp-ntpv5-05
Published: 25 August 2022
Intended Status: Standards Track
Expires: 26 February 2023
Authors: M. Lichvar
Red Hat

Network Time Protocol Version 5

Abstract

This document describes the version 5 of the Network Time Protocol (NTP).

Status of This Memo

This Internet-Draft will expire on 26 February 2023.

Copyright Notice

Table of Contents

1. Introduction

Network Time Protocol (NTP) is a protocol which enables computers to synchronize their clocks over network. Time is distributed from primary time servers to clients, which can be servers for other clients, and so on. Clients can use multiple servers simultaneously.

NTPv5 is similar to NTPv4 [RFC5905]. The main differences are:

1. The protocol specification (this document) describes only the on-wire protocol. Filtering of measurements, security mechanisms, source selection, clock control, and other algorithms, are out of scope.

2. For security reasons, NTPv5 drops support for the symmetric active, symmetric passive, broadcast, control, and private modes. The symmetric and broadcast modes are vulnerable to replay attacks. The control and private modes can be exploited for denial-of-service traffic amplification attacks. Only the client and server modes remain in NTPv5.

3. Timestamps are clearly separated from values used as cookies.

4. NTPv5 messages can be extended only with extension fields. The MAC field is wrapped in an extension field.

5. Extension fields can be of any length, even indivisible by 4, but are padded to a multiple of 4 octets. Extension fields specified for NTPv4 are compatible with NTPv5.

6. NTPv5 adds support for other timescales than UTC.

7. The NTP era number is exchanged in the protocol, which extends the unambiguous interval of the client from 136 years to about 35000 years.

8. NTPv5 adds a new measurement mode to provide clients with more accurate transmit timestamps.

9. NTPv5 works with sets of reference IDs to prevent synchronization loops over multiple hosts.

10. Resolution of the root delay and root dispersion fields is improved.

11. Clients don't leak information about their clock (e.g. timestamps).

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Basic Concepts

The distance to the reference time sources in the hierarchy of servers is called stratum. Primary time servers, which are synchronized to the reference clocks, are stratum 1, their clients are stratum 2, and so on.

Root delay measures the total delay on the path to the reference time source used by the primary time server. Each client on the path adds to the root delay the NTP delay measured to the server it considers best for synchronization. The delay includes network delays and any delays between timestamping of NTP messages and their actual reception and transmission. Half of the root delay estimates the maximum error of the clock due to asymmetries in the delay.

Root dispersion estimates the maximum error of the clock due to the instability of the clocks on the path and instability of NTP measurements. Each server on the path adds its own dispersion to the root dispersion. Different clock models can be used. In a simple model, the clock can have a constant dispersion rate, e.g. 15 ppm as used in NTPv4.

The sum of the root dispersion and half of the root delay is called root distance. It is the estimated maximum error of the clock, taking into account asymmetry in delay and stability of clocks and measurements.

Servers have randomly generated reference IDs to prevent synchronization loops.

3. Data Types

NTPv5 uses few different data types. They are all in the network order. Beside signed and unsigned integers, it has also the following fixed-point types:

time16

A 16-bit fixed-point type containing values in seconds. It has 1 signed integer bit (i.e. it is just the sign) and 15 fractional bits. The minimum value is the fraction -32767/32768 (almost -1 second), the maximum value is 32767/32768 (almost 1 second), and the resolution is about 30 microseconds. The type has a special value of 0x8000, which indicates an unknown value.

time32

A 32-bit fixed-point type containing values in seconds. It has 4 unsigned integer bits and 28 fractional bits. The maximum value is 16 seconds and the resolution is about 3.7 nanoseconds. Note that this is different than the 32-bit time format in NTPv4.

timestamp64

A 64-bit fixed-point type containing timestamps. It has 32 signed integer bits and 32 fractional bits. It spans an interval of about 136 years and has a resolution of about 0.23 nanoseconds. It can be used in different timescales. In the UTC timescale it is the number of SI seconds since 1 Jan 1972 plus 2272060800, excluding leap seconds. Timestamps in the TAI timescale are the same except they include leap seconds and extra 10 seconds for the original difference between TAI and UTC in 1972, when leap seconds were introduced. One interval covered by the type is called an NTP era. The era starting at the epoch is era number 0, the following era is number 1, and so on.

Some fields use a logarithmic scale, where an 8-bit signed integer represents the rounded log2 value of seconds. For example, a log2 value of 4 is 2 to the power of 4 (16 seconds), or a log2 value of -2 is 2 to the power of -2 (0.25 seconds).

4. Message Format

NTPv5 servers and clients exchange messages as UDP datagrams. Clients send requests to servers and servers send them back responses. The format of the UDP payload is shown in Figure 1.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|LI | VN  |Mode | Scale |Stratum|     Poll      |  Precision    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Flags     |      Era      |        Timescale Offset       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Root Delay                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Root Dispersion                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                        Server Cookie (64)                     +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                        Client Cookie (64)                     +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                      Receive Timestamp (64)                   +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                      Transmit Timestamp (64)                  +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
.                                                               .
.                    Extension Field 1 (variable)               .
.                                                               .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
.                                                               .
.                                                               .
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
.                                                               .
.                    Extension Field N (variable)               .
.                                                               .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1: Format of NTPv5 messages

Each NTPv5 message has a header containing the following fields:

Leap indicator (LI)

A 2-bit field which can have the following values: 0 (normal), 1 (leap second inserted at the end of the month), 2 (leap second deleted at the end of the month), 3 (not synchronized). The values 1 and 2 are set at most 14 days in advance before the leap second. In requests it is always 0.

Version Number (VN)

A 3-bit field containing the value 5.

Mode

A 3-bit field containing the value 3 (request) or 4 (response).

Scale

A 4-bit identifier of the timescale. In requests it is the requested timescale. In responses it is the timescale of the receive and transmit timestamps. Defined values are:

0: UTC

1: TAI

2: UT1

3: Leap-smeared UTC

Stratum

A 4-bit field containing the stratum of the server. Primary time servers have a stratum of 1, their clients have a stratum of 2, and so on. The value of 0 indicates an unknown or infinite stratum. In requests it is always 0.

Poll

An 8-bit signed integer containing the polling interval as a rounded log2 value in seconds. In requests it is the current polling interval. In responses it is the minimum allowed polling interval.

Precision

An 8-bit signed integer containing the precision of the timestamps included in the message as a rounded log2 value in seconds. In requests, which don't contain any timestamps, it is always 0.

Flags

An 8-bit integer that can contain the following flags:

0x1: Unknown leap

In requests it is 0. In responses a value of 1 indicates the server does not have a time source which provides information about leap seconds and the client should interpret the Leap

Indicator as having only two possible values: synchronized (0) and not synchronized (3).

0x2: Interleaved mode

In requests a value of 1 is a request for a response in the interleaved mode. In responses a value of 1 indicates the response is in the interleaved mode.

Era

An 8-bit unsigned NTP era number corresponding to the receive timestamp. In requests it is always 0.

Timescale Offset

A 16-bit value specific to the selected timescale, which is referenced to the receive timestamp. In requests it is always 0.

- In the UTC (0) and TAI (1) timescales it is the TAI-UTC offset (TAI minus UTC) as a signed integer, or 0x8000 if unknown.
- In the UT1 timescale (2) it is the UT1-UTC offset (UT1 minus UTC) using the time16 type (0x8000 if unknown).
- In the leap-smeared UTC (3), it is the current offset between the leap smeared time and UTC (former minus latter) using the time16 type (0x8000 if unknown).

Root Delay

A field using the time32 type. In responses it is the server's root delay. In requests it is always 0.

Root Dispersion

A field using the time32 type. In responses it is the server's root dispersion. In requests it is always 0.

Server Cookie

A 64-bit field containing a number generated by the server which enables the interleaved mode. In requests it is 0, or a copy of the server cookie from the last response.

Client Cookie

A 64-bit field containing a random number generated by the client. Responses contain a copy of the field from the corresponding request, which allows the client to verify that the responses are valid responses to the requests.

Receive Timestamp

A field using the timestamp64 type. In requests it is always 0. In responses it is the time when the request was received. The timestamp corresponds to the end of the reception.

Transmit Timestamp

A field using the timestamp64 type. In requests it is always 0. In responses it is the beginning of the transmission of a response to the client. Which response it refers to depends on the selected mode (basic or interleaved). See Measurement Modes (Section 6) for detail.

The header has 48 octets, which is the minimum length of a valid NTPv5 message. A message can contain zero, one, or multiple extension fields. The maximum length is not specified, but the length is always divisible by 4.

5. Extension Fields

The format of NTPv5 extension fields is shown in Figure 2.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Type              |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
.                                                               .
.                        Data (variable)                        .
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: Format of NTPv5 extension fields

Each extension field has a header which contains a 16-bit type and 16-bit length. The length is in octets and it includes the header. The minimum length is 4, i.e. an extension field does not have to contain any data. If the length is not divisible by 4, the extension field is padded with zeroes to the smallest multiple of 4 octets.

If a request contains an extension field, the server MUST include this extension field in the response unless the specification of the extension field states otherwise, or the server does not support the extension field. A client can interpret the absence of an expected extension field in a response as an indication that the server does not support the extension field.

Extension fields specified for NTPv4 can be included in NTPv5 messages as specified for NTPv4.

The rest of this section describes new extension fields specified for NTPv5. Clients are not required to use or support any of these extension fields, but servers are required to support some extension fields.

5.1. Padding Extension Field

This field is used by servers to pad the response to the same length as the request if the response does not contain all requested extension fields, or some have a variable length. It can have any length.

This field MUST be supported on server.

5.2. MAC Extension Field

This field authenticates the NTPv5 message with a symmetric key. Implementations SHOULD use the MAC specified in RFC8573 [RFC8573]. The extension field MUST be the last extension field in the message unless an extension field is specifically allowed to be placed after a MAC or another authenticator field.

5.3. Reference IDs Request and Response Extension Fields

Each NTPv5 server has a randomly generated 120-bit reference ID. The extension fields described in this section are used to exchange sets of reference IDs in order to detect synchronization loops, i.e. when a client is synchronizing (directly or indirectly) to one of its own clients.

As each client can be synchronized to an unlimited number of servers (and there can be up to 15 strata of servers), the reference IDs are exchanged as a Bloom filter instead of a list to limit the amount of data that needs to be exchanged.

The Bloom filter is an array of 4096 bits. When empty, all bits are zero. To add a reference ID to the filter, the 120-bit value of the reference ID is split into 10 12-bit values and the bits of the array at the 10 positions given by the 12-bit values are set to one.

A server maintains a copy of the filter for each server it is using as an NTP client. The filter provided by the server to clients is the union of the filters (using the bitwise OR operation) of the server's sources selected for synchronization and the server's own reference ID.

If the server uses a previous version of NTP for some of its sources, the reference IDs added to the filter are generated from their IP addresses as the first 120 bits of the MD5 [RFC1321] sum of the address.

A client checking whether the server's set of reference IDs contains the client's own reference ID checks whether the bits at the 10 positions corresponding to the 12-bit values from the reference ID are all set to one.

When a client which serves time to other clients detects a synchronization loop with one of its servers, it SHOULD stop using the server for synchronization. When the client's reference ID is no longer detected in the server's filter, it SHOULD wait for a random number of polling intervals (e.g. between 0 and 4) before selecting the server again. The random delay helps with stabilization of the selection in longer loops.

False positives are possible. The probability of a collision grows with the number of reference IDs in the filter. With 26 reference IDs it is about 1e-12. With 118 IDs it is about 1e-6. The client MAY avoid selecting a server which has too many bits set in the filter (e.g. more than half) to reduce the probability of the collision for its own clients. A client which detected a synchronization loop MAY change its own reference ID to limit the duration of the potential collision.

The filter can be exchanged as a single 512-octet array, or it can be exchanged in smaller chunks over multiple NTP messages, making them shorter, but delaying the detection of the synchronization loop.

The request extension field specifies the offset of the requested chunk in the filter as a number of octets. The requested length of the chunk is given by the length of the extension field. The response extension field MUST have the same length as the request extension field. If the request contains an invalid offset, the extension field MUST be ignored.

The client SHOULD use requests of a constant length for the association to avoid adding a variation to the measured NTP delay.

The format of the Reference IDs Request is shown in Figure 3.

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |             Type              |            Length             |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |            Offset             |                               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
 .                                                               .
 .                        Padding (variable)                     .
 .                                                               .
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3: Format of Reference IDs Request Extension Field

The format of the Reference IDs Response is shown in Figure 4.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Type              |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
.                                                               .
.                  Bloom filter chunk (variable)                .
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: Format of Reference IDs Response Extension Field

These fields MUST be supported on server.

5.4. Server Information Extension Field

This field provides clients with information about which NTP versions are supported by the server, as a minimum and maximum version. The extension field has a fixed length of 8 octets. In requests, all data fields of the extension are 0.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Type              |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Min. Version  | Max. Version  |            Reserved           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5: Format of Server Information Extension Field

This field MUST be supported on server.

5.5. Correction Extension Field

Processing and queueing delays in network switches and routers may be a significant source of jitter and asymmetry in network delay, which has a negative impact on accuracy and stability of clocks synchronized by NTP. A solution to this problem is defined in the Precision Time Protocol (PTP) [IEEE1588], which is a different protocol for synchronization of clocks in networks. In PTP a special type of switch or router, called a Transparent Clock (TC), updates a correction field in PTP messages to account for the time messages spend in the TC. This is accomplished by timestamping the message at the ingress and egress ports, taking the difference to determine time in the TC and adding this to the Delay Correction. Clients can account for the accumulated Delay Correction to determine a more accurate clock offset.

The NTPv5 Delay Correction has the same format as the PTP correctionField to make it easier for manufacturers of switches and routers to implement NTP corrections. The format of the Correction Extension Field is shown in Figure 6.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Type              |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                       Origin Correction                      +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Origin path ID        |            Reserved           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                       Delay Correction                        +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Delay Path ID        |     Checksum complement       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6: Format of Correction Extension Field

Field Type

The type which identifies the Correction extension field (value TBD).

Length

The length of the extension field, which is 28 octets.

Origin Correction

A field which contains a copy of the accumulated delay correction from the request packet in the NTP exchange.

Origin ID

A field which contains a copy of the final path ID from the request packet in the NTP exchange.

Reserved

16 bit reserved for future specification by the IETF. Transmit with all zeros.

Delay Correction

A signed fixed-point number of nanoseconds with 48 integer bits and 16 binary fractional bits, which represents the current correction of the network delay that has accumulated for this packet on the path from the source to the destination. The format of this field is identical to the PTP correctionField.

Path ID

A 16-bit identification number of the path where the delay correction was updated.

Checksum Complement

A field which can be modified in order to keep the UDP checksum of the packet valid. This allows the UDP checksum to be transmitted before the Correction Field is received and modified. The same field is described in RFC 7821 [RFC7821].

A correction capable client SHALL transmit the request with the Origin Correction, Origin ID, Delay Correction and Path ID fields filled with all zeros.

Network nodes, such as switches and routers, that are NTP corrections capable SHALL add the difference between the beginning of an NTP message retransmission and the end of the message reception to the received Delay Correction value, and update this field. Note that this time difference might be negative, for example in a cut-through switch. If the packet is transmitted at the same speed as it was received and the length of the packet does not change (e.g. due to adding or removing a VLAN tag), the beginning and end of the interval may correspond to any point of the reception and transmission as long as it is consistent for all forwarded packets of the same length. If the transmission speed or length of the packet is different, the beginning and end of the interval SHOULD correspond to the end of the reception and beginning of the transmission respectively. Both timestamps MUST be based on the same clock. This clock does not need to be synchronized as long as the frequency is accurate enough such that resulting time difference estimation errors are acceptable to the precision required by the application.

If a network node updates the delay correction, it SHOULD also add the identification numbers of the incoming and outgoing port to the path ID. Path ID values can be used by clients to determine if the ntp request and response messages are likely to have traversed the same network path.

If a network node modified any field of the extension field, it MUST update the checksum complement field in order to keep the current UDP checksum valid, or update the UDP checksum itself.

The server SHALL write the received Delay Correction value in the origin correction field of the response message, and the received path ID value in the origin ID field. The server SHALL set the Delay Correction field and Path ID fields to all zeros

5.6. Reference Timestamp Extension Field

This fields contains the time of the last update of the clock. It has a fixed length of 12 octets. In requests, the timestamp is always 0.

(Is this really needed? It was mostly unused in NTPv4.)

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Type              |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                      Reference Timestamp (64)                 |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 7: Format of Reference Timestamp Extension Field

5.7. Monotonic Timestamp Extension Field

When a clock is synchronized to a time source, there is a compromise between time (phase) accuracy and frequency accuracy, because the frequency of the clock has to be adjusted to correct time errors that accumulate due to the frequency error (e.g. caused by changes in the temperature of the crystal). Faster corrections of time can minimize the time error, but increase the frequency error, which transfers to clients using that clock as a time source and increases their frequency and time errors. This issue can be avoided by transferring time and frequency separately using different clocks.

The Monotonic Timestamp Extension Field contains an extra receive timestamp with a 32-bit epoch identifier captured by a clock which does not have corrected phase and can better transfer frequency than the clock which captures the receive and transmit timestamps in the header. The extension field has a constant length of 16 octets. In requests, the counter and timestamp are always 0.

The epoch identifier is a random number which is changed when frequency transfer needs to be restarted, e.g. due to a step of the clock.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Type              |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Epoch ID                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                  Monotonic Receive Timestamp (64)             |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 8: Format of Monotonic Timestamp Extension Field

The client can determine the frequency-transfer offset from the time-transfer offset and difference between the two receive timestamps in the response. It can use the frequency-transfer offset to better control the frequency of its clock, avoiding the frequency error in the server's time-transfer clock.

6. Measurement Modes

An NTPv5 client needs four timestamps to measure the offset and delay of its clock relative to the server's clock:

1. T1 - client's transmit timestamp of a request

2. T2 - server's receive timestamp of the request

3. T3 - server's transmit timestamp of a response

4. T4 - client's receive timestamp of the response

The offset, delay and dispersion are calculated as:

*offset = ((T2 + T3) - (T4 + T1) + (Cd - Co)) / 2

*delay = |(T4 - T1) - (T3 - T2) - (Cd + Co)|

*dispersion = |T4 - T1| * DR

where

*T1, T2, T3, T4 are the receive and transmit timestamps of a request and response

*Co is the Origin Correction from the Correction Extension Field if present in the response and has acceptable values, zero otherwise

*Cd is the Delay Correction from the Correction Extension Field if present in the response and has acceptable values, zero otherwise

*DR is the client's dispersion rate

The client can make measurements in the basic mode, or interleaved mode if supported on the server. In the basic mode, the transmit timestamp in the server response corresponds to the message which contains the timestamp itself. In the interleaved mode it corresponds to a previous response identified by the server cookie. The interleaved mode enables the server to provide the client with a more accurate transmit timestamp which is available only after the response was formed or sent.

An example of cookies and timestamps in an NTPv5 exchange using the basic mode is shown in Figure 9.

```
Server   t2   t3               t6   t7              t10  t11
    -----+----+----------------+----+----------------+----+-----
        /      \              /      \              /      \
Client /        \            /        \            /        \
    --+----------+----------+----------+----------+----------+--
      t1         t4         t5         t8         t9        t12

    +----+    +----+      +----+    +----+      +----+    +----+
SC  | 0  |    | s1 |      | 0  |    | s2 |      | 0  |    | s3 |
CC  | c1 |    | c1 |      | c2 |    | c2 |      | c3 |    | c3 |
Rx  | 0  |    | t2 |      | 0  |    | t6 |      | 0  |    |t10 |
Tx  | 0  |    | t3 |      | 0  |    | t7 |      | 0  |    |t11 |
    +----+    +----+      +----+    +----+      +----+    +----+
```

Figure 9: Cookies and timestamps in basic mode

From the three exchanges in this example, the client would use the the following sets of timestamps:

*(t1, t2, t3, t4)

*(t5, t6, t7, t8)

*(t9, t10, t11, t12)

For NTPv4, the interleaved mode is described in NTP Interleaved Modes [I-D.ietf-ntp-interleaved-modes]. The difference between the NTPv5 and NTPv4 interleaved modes is that in NTPv5 it is enabled with a flag and the previous transmit timestamp on the server is identified by the server cookie instead of the receive timestamp.

An example of an NTPv5 exchange using the interleaved mode is shown in Figure 10. The messages in the basic and interleaved mode are indicated with B and I respectively. The timestamps t3' and t11' correspond to the same transmissions as t3 and t11, but they may be less accurate. The first exchange is in the basic mode followed by a second exchange in the interleaved mode. For the third exchange, the client request is in the interleaved mode, but the server response is in the basic mode, because the server no longer had the timestamp t7 (e.g. it was dropped to save timestamps for other clients using the interleaved mode).

```
Server   t2   t3               t6   t7                t10  t11
    -----+----+----------------+----+----------------+----+-----
        /      \              /      \              /      \
Client /        \            /        \            /        \
    --+----------+----------+----------+----------+----------+--
      t1         t4         t5         t8         t9        t12

Mode: B         B           I         I           I         B
    +----+    +----+      +----+    +----+      +----+    +----+
SC  | 0  |    | s1 |      | s1 |    | s2 |      | s2 |    | s3 |
CC  | c1 |    | c1 |      | c2 |    | c2 |      | c3 |    | c3 |
Rx  | 0  |    | t2 |      | 0  |    | t6 |      | 0  |    |t10 |
Tx  | 0  |    | t3'|      | 0  |    | t3 |      | 0  |    |t11'|
    +----+    +----+      +----+    +----+      +----+    +----+
```

Figure 10: Cookies and timestamps in interleaved mode

From the three exchanges in this example, the client would use the following sets of timestamps:

- (t1, t2, t3', t4)

- (t1, t2, t3, t4) or (t5, t6, t3, t4)

- (t9, t10, t11', t12)

7. Client Operation

An NTPv5 client can use one or multiple servers. It has a separate association with each server. It makes periodic measurements of its offset and delay to the server. It can filter the measurements and compare measurements from different servers to select and combine the best servers for synchronization. It can adjust its clock in order to minimize its offset and keep the clock synchronized. These algorithms are not specified in this document.

The polling interval can be adjusted for the network conditions and stability of the clock. When polling a public server on Internet, the client SHOULD use at least a polling interval of 64 seconds, increasing up to at least 1024 seconds.

Each successful measurement provides the client with an offset, delay and dispersion. When combined with the server's root delay and dispersion, it gives the client an estimate of the maximum error.

On each poll, the client:

1. Generates a new random cookie.

2. Formats a request with necessary extension fields and the fields in the header all zero except:

   *Version is set to 5.

   *Mode is set to 3.

   *Scale is set to the timescale in which the client wants to operate.

   *Poll is set to the rounded log2 value of the current client's polling interval in seconds.

   *Flags are set according to the requested mode. The interleaved mode flag requests the server to save the transmit timestamp of the response and provide the transmit timestamp of a previous response corresponding to the server cookie (if not zero).

   *Server cookie is set only in the interleaved mode. It is set to the server cookie from the last valid response, or zero if no such response was received yet or the transmit timestamp of that response would no longer be useful to the client (e.g. after missing too many responses).

   *Client cookie is set to the newly generated cookie.

3. Sends the request to the server to the UDP port 123 and captures a transmit timestamp.

4. Waits for a valid response from the server and captures a receive timestamp. A valid response has version 5, mode 4, client cookie equal to the cookie from the request, and passes authentication if enabled. The client MUST ignore all invalid responses and accept at most one valid response.

5. Checks whether the response is usable for synchronization of the clock. Such a response has a leap indicator not equal to 3, stratum between 0 and 16, root delay and dispersion both

smaller than a specific value, e.g. 16 seconds, and timescale equal to the requested timescale. If the response is in a different timescale, the client can switch to the provided timescale, convert the timestamps if the offset between the timescales is provided or known, or drop the response.

6. Saves the server's receive and transmit timestamps. If the client internally counts seconds using a type wider than 32 bits, it SHOULD expand the timestamps with the provided NTP era.

7. Calculates the offset, delay, and dispersion.

A client which operates as a server for other clients MUST include the Reference IDs Request Extension Field in its requests in order to track reference IDs of its sources. If the server's set of reference IDs contains the client's own reference ID, it SHOULD not select the server for synchronization to avoid a synchronization loop.

8. Server Operation

A server receives requests on the UDP port 123. The server MUST support measurements in the basic mode. It MAY support the interleaved mode.

For the basic mode the server does not need to keep any client-specific state. For the interleaved mode it needs to save transmit timestamps and be able to identify them by a cookie.

The server maintains its leap indicator, stratum, root delay, and root dispersion:

- Leap indicator MUST be 3 if the clock is not synchronized or its maximum error cannot be estimated with the root delay and dispersion. Otherwise, it MUST be 0, 1, 2, depending on whether a leap second is pending in the next 14 day and, if it is, whether it will be inserted or deleted.

- Stratum SHOULD be one larger than stratum of the best server it uses for its own synchronization.

- Root delay SHOULD be the best server's root delay in addition to the measured delay to the server.

- Root dispersion SHOULD be the best server's root dispersion in addition to an estimate of the maximum drift of its own clock since the last update of the clock.

The server has a randomly generated 120-bit reference ID. It MUST track reference IDs of its servers in order to be able to respond with a Reference IDs Response Extension Field.

For each received request, the server:

1. Captures a receive timestamp.
2. Checks the version in the request. If it is not equal to 5, it MUST either drop the request, or handle it according to the specification corresponding to the protocol version.
3. Drops the request if the format is not valid, mode is not 3, or authentication fails with the MAC Extension Field or another authenticator which does not have a specified response for failed authentication. The server MUST ignore unknown extension fields.
4. Server forms a response with requested extension fields and sets the fields in the header as follows:

   - Leap Indicator, Stratum, Root delay, and Root dispersion, are set to the current server's values.

   - Version is set to 5.

   - Scale is set to the client's requested timescale if it is supported by the server. If not, the server SHOULD respond in any timescale it supports.

   - The flags are set as follows:

     Unknown leap is set if the server does not know if a leap second is pending in the next 14 days, i.e. it has no source providing information about leap seconds.

     Interleaved mode is set if the interleaved mode was requested and a response in the interleaved mode is possible (i.e. a transmit timestamp is associated with the server cookie).

   - Era is set to the NTP era of the receive timestamp.

   - Timescale Offset is set to the timescale-specific offset, or 0x8000 if unknown.

   - Server Cookie is set when the interleaved mode is requested and it is supported by the server, even if the response cannot be in the requested mode due to the request having an unknown or zero server cookie. The cookie identifies a more accurate transmit timestamp of the response, which can be retrieved by the client later with another request. The cookie generation is implementation-specific.

- Client Cookie is set to the Client Cookie from the request.

- Receive Timestamp is set to the server's receive timestamp of the request.

- Transmit Timestamp is set to a value which depends on the measurement mode. In the basic mode it is the server's current time when the message if formed. In the interleaved mode it is the transmit timestamp of the previous response identified by the server cookie in the request, captured at some point after the message was formed.

5. Adds the Padding Extension field if necessary to make the length of the response equal to the length of the request.

6. Drops the response if it is longer than the request to prevent traffic amplification.

7. Sends the response.

8. Saves the transmit timestamp and server cookie, if the interleaved mode was requested and is supported by the server.

9. Network Time Security with NTPv5

The Network Time Security [RFC8915] mechanism uses the NTS-KE protocol to establish keys and negotiate the next protocol. NTPv5 is added as a new protocol to the Network Time Security Next Protocols Registry, which can be negotiated by NTPv5 clients and servers supporting NTS.

No new NTS-KE records are specified for NTPv5. The records that were specified for NTPv4 (i.e. NTPv4 New Cookie, NTPv4 Server Negotiation, and NTPv4 Port Negotiation) are reused for NTPv5.

The NTS extension fields specified for NTPv4 are compatible with NTPv5. No new extension fields are specified.

10. NTPv5 Negotiation in NTPv4

NTPv5 messages are not compatible with NTPv4, even if they do not contain any extension fields. Some widely used NTPv4 implementations are known to ignore the version and interpret all requests as NTPv4. Their responses to NTPv5 requests have a zero client cookie, which means they fail the client's validation and are ignored.

The implementations are also known to not respond to requests with an unknown extension field, which prevents an NTPv4 extension field to be specified for NTPv5 negotiation. Instead, the reference timestamp field in the NTPv4 header is reused for this purpose.

An NTP server which supports both NTPv4 and NTPv5 SHOULD check the reference timestamp in all NTPv4 client requests. If the reference timestamp contains the value 0x4E5450354E545035 ("NTP5NTP5" in ASCII), it SHOULD respond with the same reference timestamp to indicate it supports NTPv5.

An NTP client which supports both NTPv4 and NTPv5, does not use NTS, and is not configured to use a particular NTP version, SHOULD start with NTPv4 and set the reference timestamp to 0x4e5450354e545035. If the server responds with the same reference timestamp, the client SHOULD switch to NTPv5. If no valid response is received for a number of requests (e.g. 8), the client SHOULD switch back to NTPv4.

11. Acknowledgements

Some ideas were taken from a different NTPv5 design proposed by Daniel Franke.

The author would like to thank Doug Arnold for his contributions and Dan Drown, Watson Ladd, Hal Murray, Kurt Roeckx, and Ulrich Windl for their suggestions and comments.

12. IANA Considerations

IANA is requested to allocate the following protocol in the Network Time Security Next Protocols Registry [RFC8915]:

Protocol ID: [[TBD]], selected by IANA from the IETF Review range

Protocol Name: Network Time Protocol version 5 (NTPv5)

Reference: [[this memo]]

13. Security Considerations

14. References

14.1. Normative References

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, DOI 10.17487/RFC1321, April 1992, <https://www.rfc-editor.org/info/rfc1321>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/

RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8573] Malhotra, A. and S. Goldberg, "Message Authentication Code for the Network Time Protocol", RFC 8573, DOI 10.17487/RFC8573, June 2019, <https://www.rfc-editor.org/info/rfc8573>.

14.2. Informative References

[I-D.ietf-ntp-interleaved-modes] Lichvar, M. and A. Malhotra, "NTP Interleaved Modes", Work in Progress, Internet-Draft, draft-ietf-ntp-interleaved-modes-07, 18 October 2021, <https://datatracker.ietf.org/api/v1/doc/document/draft-ietf-ntp-interleaved-modes/>.

[IEEE1588] Institute of Electrical and Electronics Engineers, "IEEE std. 1588-2019, "IEEE Standard for a Precision Clock Synchronization for Networked Measurement and Control Systems."", November 2019, <https://www.ieee.org>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

[RFC7821] Mizrahi, T., "UDP Checksum Complement in the Network Time Protocol (NTP)", RFC 7821, DOI 10.17487/RFC7821, March 2016, <https://www.rfc-editor.org/info/rfc7821>.

[RFC8915] Franke, D., Sibold, D., Teichel, K., Dansarie, M., and R. Sundblad, "Network Time Security for the Network Time Protocol", RFC 8915, DOI 10.17487/RFC8915, September 2020, <https://www.rfc-editor.org/info/rfc8915>.

Author's Address

Miroslav Lichvar
Red Hat
Purkynova 115
612 00 Brno
Czech Republic

Email: mlichvar@redhat.com