

NTP WG  
Internet-Draft  
Intended status: Standards Track  
Expires: 6 December 2021

H. Gerstung  
M. Rohde  
D. Arnold  
Meinberg  
4 June 2021

Network Time Security for the Unicast Mode of the Precision Time  
Protocol  
draft-gerstung-nts4uotp-03

Abstract

This memo specifies the application of Network Time Security, a mechanism for using Transport Layer Security (TLS) and Authenticated Encryption with Associated Data (AEAD) to provide cryptographic security for the unicast mode of the Precision Time Protocol.

It is based on the 'Network Time Security for the Network Time Protocol' document RFC8915 and re-uses most of its mechanisms for providing a secure and robust key exchange solution for unicast PTP. Due to the different modes of operation, additional steps are required to secure unicast PTP communication between the PTP clients and unicast PTP servers. In addition to defining the new record types and other required values to allow the utilization of the NTS key exchange sub protocol, there are a number of additional protocol enhancements and server-side requirements which are defined in this memo.

NOTE

This document is work in progress

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 6 December 2021.

#### Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

#### Table of Contents

1. Introduction . . . . .	3
1.1. Terminology . . . . .	3
2. Objectives . . . . .	3
3. Application of the NTS protocol to PTP . . . . .	4
3.1. Phase 1: NTS-KE Phase . . . . .	7
3.2. Phase 2: PTP Unicast Transmission Negotiation Phase . . . . .	7
3.3. Phase 3: PTP Unicast Packet Transmission Phase . . . . .	12
4. New NTS record types . . . . .	13
4.1. Cookie for Unicast PTP . . . . .	13
4.2. Unicast PTP Server Negotiation . . . . .	13
5. The PTP client Table . . . . .	14
6. The NTS_TLV . . . . .	14
7. IANA Considerations . . . . .	17
8. Security Considerations . . . . .	17
8.1. Threat Model . . . . .	17
8.2. General Security Features . . . . .	18
9. Delay Attacks . . . . .	19
10. Acknowledgements . . . . .	19
11. References . . . . .	19
11.1. Normative References . . . . .	19
11.2. Informative References . . . . .	20
Authors' Addresses . . . . .	20

## 1. Introduction

This memo specifies Network Time Security for unicast mode of the Precision Time Protocol (PTP). It is based on [RFC8915] and applies the key exchange mechanism described there to PTP. The Precision Time Protocol is standardized in [IEEE1588] and offers a number of different modes and mappings to communication protocols. The security mechanisms described here provide a way to secure the unicast mode of PTP as specified in sub clause 16.1 of [IEEE1588].

The PTP integrated security mechanism has been specified in sub clause 16.14 of [IEEE1588] and introduces an AUTHENTICATION TLV that carries all necessary information to enable the receiver of a PTP message to verify its integrity. Although two different approaches are described in that sub clause (immediate and delayed security processing), NTS4UPTP only uses the immediate security processing.

In addition to sub clause 16.14, Annex P of [IEEE1588] provides additional explanation and description of PTP security. It is stated there that for key management it is assumed that a separate mechanism outside the context of PTP is used. In P.2.1.2 the document clearly states that this assumption was made in relation to the security mechanism described in 16.14 and it goes on to discuss some Key management options and it names both manual and automatic key management as possible approaches.

This memo describes a way to use the automatic key exchange mechanism as defined in [RFC8915] as the key management for unicast PTP. The NTS-KE protocol has clearly been designed to support using it for multiple time synchronization protocols and this document is utilizing this support.

### 1.1. Terminology

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 [RFC2119].

## 2. Objectives

The objectives of NTS are defined in [RFC8915] and, with some exceptions, apply to NTS4UPTP as well.

- \* Identity: Through the use of a X.509 public key infrastructure, implementations can cryptographically establish the identity of the parties they are communicating with.

- \* Authentication: Implementations can cryptographically verify that any time synchronization packets are authentic, i.e., that they were produced by an identified party and have not been modified in transit.
- \* Replay prevention: clients and servers can detect when a received packet is a replay of a previous packet.
- \* Request-response consistency: clients can verify that a unicast PTP packet received from a server was sent in response to a particular request from the client.
- \* Non-amplification: Implementations (especially server implementations) can avoid acting as distributed denial-of-service (DDoS) amplifiers by never responding to a packet with one or more packets creating more traffic than the initiating packet.
- \* Scalability: Server implementations can serve large numbers of clients.
- \* Performance: NTS must not significantly degrade the quality of the time transfer. The encryption and authentication used when actually transferring time should be lightweight.

The following objectives of [RFC8915] are not met in this proposal:

- \* Confidentiality: Basic time synchronization data is considered nonconfidential and sent in the clear. Despite this, NTS4NTP includes support for encrypting NTP extension fields. NTS4UPTP does not offer this kind of support as it is not considered useful or required for unicast PTP implementations.
- \* Unlinkability: For mobile clients, NTS4NTP does not leak any information additional to NTP which would permit a passive adversary to determine that two packets sent over different networks came from the same client. This objection cannot be achieved by unicast PTP because the protocol requires the server to keep a state for all its clients. It is also not considered a requirement in most applications where unicast PTP is deployed.

### 3. Application of the NTS protocol to PTP

Unlike NTP [RFC5905] which uses a request-response communication approach, the unicast mode of PTP is applying a subscription based model. Although [IEEE1588] allows unicast operation without negotiation this is rarely used. For this reason only unicast PTP with negotiation is considered. A PTP client (PTP Ordinary Clock, or synchronizing port of a PTP unicast Boundary Clock) sends a request

for the transmission of packets to a PTP instance offering such a service. This could be a PTP unicast Grandmaster instance or a PTP boundary clock. Note that [IEEE1588] allows PTP Ports in a states other than master to accept unicast message grant requests and act as a unicast PTP master. This option can be used for monitoring purposes. In this memo we treat all unicast associations in the same way regardless of whether it is for purposes of time transfer or monitoring, and refer to the port that grants message contracts as a PTP server.

A PTP server that receives a message grant request will then either accept the request or deny it, for example based on capacity considerations or its own operational state. This sub protocol of IEEE 1588 is called unicast message negotiation, an optional feature defined in sub clause 16.1 of [IEEE1588]. Both the PTP client and the PTP server granting a request can cancel a subscription (referred to as contract in PTP) after it has been granted and each contract includes a duration after which the PTP instance stops sending packets automatically if the PTP client did not request a new contract before the old contract ended.

This results in a 3-phase approach for PTP:

- \* Phase 1: NTS-KE Phase (Section 3.1)
- \* Phase 2: PTP Unicast Transmission Negotiation Phase (Section 3.2)
- \* Phase 3: PTP Unicast Packet Transmission Phase (Section 3.3)

In a typical use-case, phase 1 is required to be performed at startup. In phase 2 the PTP client and the PTP server will negotiate the transmission of PTP messages which will then be delivered by the PTP server in phase 3. Whenever phase 3 ends, the PTP client must re-run phase 2 to re-request more packets. Typically, PTP clients will re-run phase 2 before the active contract ends, i.e. they request a new transmission contract from the PTP server with a new duration before the active contract expires in order to secure a continuous flow of messages from the PTP server.

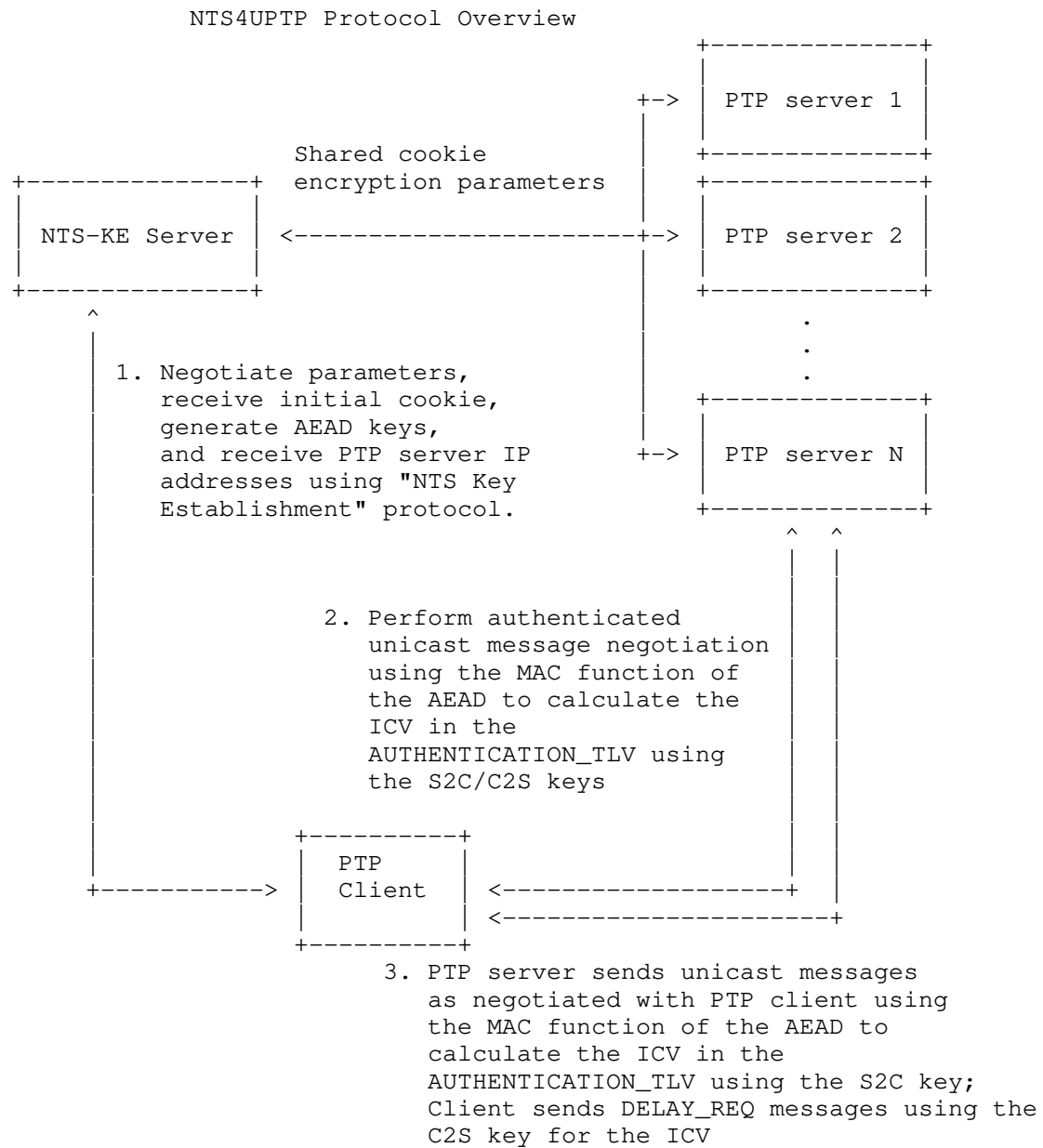


Figure 1: Overview of High-Level Interactions in NTS4UPTP

Phase 1 only needs to be re-run to avoid that the key lifetime expires, the PTP server stops responding or does not accept the cookie presented by the PTP client for any reasons.

### 3.1. Phase 1: NTS-KE Phase

In the NTS-KE Phase (Phase 1), the PTP client connects to the NTS-KE server via a secure TLS channel. Two keys are generated based on the TLS data exchange, referred to as Server-to-Client key (S2C key) and Client-to-Server key (C2S key). The NTS-KE server creates a cookie for the PTP client, which contains those two keys, the AEAD algorithm used and a Nonce. The cookie is secured by encrypting this information with a master key K. The master key is generated by a PTP server and sent to the KE server via a proprietary mechanism. The client therefore cannot decrypt a cookie as it does not know the master key. After receiving the cookies and establishing the C2S and S2C keys, the TLS connection is closed. This is identical to the NTS-KE phase as defined in [RFC8915] with the exception that the NTS-KE record type "next-protocol" points to PTP instead of NTP and two new NTS record types are introduced:

- \* "Cookie for Unicast PTP" provides the client with the cookie required to establish a valid NTS connection with the PTP server. This NTS record type is defined in Section 4.1.
- \* "PTP server Negotiation" tells the client which PTP server it MUST use. This NTS record type is defined in Section 4.2.

The MAC function of the AEAD algorithm will be used to create/calculate the ICV in the AUTHENTICATION\_TLV as defined in subclause 16.14 of [IEEE1588] (Integrated PTP Security).

For providing the cookie, an NTS-KE server MUST use a new NTS record type (New Cookie for Unicast PTP), which is identical to the NTS record type 5 (New Cookie for NTPv4) as described in [RFC8915].

The S2C key will be used in Phase 2 and 3 to calculate the ICV of the AUTHENTICATION\_TLV for all PTP messages sent from the PTP server to the PTP client. The C2S key will be used in Phase 2 and 3 to calculate the ICV of the AUTHENTICATION\_TLV for all PTP messages sent from the PTP client to the PTP server.

After the PTP client successfully completed Phase 1, it can enter Phase 2 by initiating the PTP unicast negotiation with the provided PTP server.

### 3.2. Phase 2: PTP Unicast Transmission Negotiation Phase

A unicast PTP client needs to establish a contract with a PTP server if it wants to receive SYNC and ANNOUNCE messages and to perform delay measurements by sending DELAY\_REQ messages to the PTP server, which responds with DELAY\_RESP messages.

The mechanism to establish these contracts between PTP client and PTP server is described in subclause 16.1 of [IEEE1588] ("Unicast message negotiation"). The basic concept requires the PTP client to send a request for each specific message type to transmit that message at a specific rate for a specific duration. The PTP server either grants the request or rejects it (for example due to capacity constraints). Each message type requires its own contract between a PTP client and a PTP server and there can only be one active contract per message type between the two nodes.

According to [IEEE1588] PTP messages can be extended by adding one or more TLVs on the end of the PTP message, and [IEEE1588] defines a number of TLVs. Here TLV stands for type-length-value. A standards development organization can also define TLVs to support specifications for extending PTP. In this case the TLV will be "ORGANIZATION\_EXTENSION\_PROPAGATE" or "ORGANIZATION\_EXTENSION\_DO\_NOT\_PROPAGATE" depending on whether a PTP Boundary Clock shall pass the TLV on or not. This kind of TLV MUST include a field for with the organizationID and a field with the organizationSubType. The latter MUST be unique among PTP TLVs defined by that organization.

The nature of unicast PTP requires that a PTP server maintains a list of PTP clients with active contracts. For NTS4UPTP, a server needs to store additional data. The storage entity required for storing the additional data is referred to as the PTPCLTABLE (Section 5) in this document.

In order to secure the PTP unicast transmission negotiation, PTP clients and servers use cookies and nonces, and protect the integrity of the PTP signaling messages with the integrated security mechanism based on the AUTHENTICATION\_TLV described in [IEEE1588]. A PTP client initially requires a cookie for the first message it sends during this phase and will receive a nonce each time the PTP server sends a response. The initial cookie for the first request of the client is provided by the NTS-KE server in the NTS-KE Phase. For each consecutive message the PTP client sends, it MUST use the nonce received in the previous message from the PTP server and send that one back in the NONCE field of the NTS\_TLV.

Due to the fact, that the S2C/C2S key pair expires, the client is forced to get new keys from the NTS-KE server. The client MAY do this at any time and MUST do it when receiving a NTS\_INVALIDKEYS response from the PTP server.

Nonces and cookies are not required in the third phase, in which the transmitted PTP messages are only secured with the AUTHENTICATION\_TLV. Packet rates in this phase can be very high, PTP



for example allows for up to 128 SYNC packets per second sent by the PTP server to a client. Due to the fact that PTP requires a client to successfully complete the negotiation phase, it is sufficient to protect the integrity of the messages in the transmission phase with the AUTHENTICATION\_TLV.

The unicast negotiation mechanism as specified in [IEEE1588] is carried out according to the standard, but with the following addition:

- \* The PTP client and the PTP server MUST add an NTS TLV and an AUTHENTICATION\_TLV to all signaling messages
- \* The NTS\_TLV (Section 6) in a message sent from the PTP client to the PTP server either carries the cookie that the client obtained during the last successfully completed NTS-KE Phase, or the last nonce provided by the PTP server in its response. Additionally the ntsMsgId MUST be set to NTS\_INIT for the first message sent to the PTP server after completing the NTS-KE phase and to NTS\_REQUEST for all further messages.
- \* The AUTHENTICATION\_TLV secures the REQUEST\_UNICAST\_TRANSMISSION\_TLV and the NTS\_TLV with an ICV which is calculated based on the MAC function of the AEAD algorithm and the C2S key that has been obtained during the NTS-KE phase
- \* All signaling messages from the PTP server to the PTP client MUST carry a new nonce and the ICV in the AUTHENTICATION\_TLV is calculated based on the S2C key that the PTP server read from the decrypted cookie in the last NTS\_INIT message from the client.

All PTP messages in the packet negotiation phase that do not carry an AUTHENTICATION\_TLV or in which the ICV is not correct MUST be ignored by the recipient.

A PTPCLTABLE entry SHALL be stored at least for as long as the S2C/C2S key pair is valid, i.e. until KEY\_PAIR\_EXP has been reached. The maximum lifetime of a key pair is defined in the configuration of the PTP server and the expiration date/time is calculated when the entry in this PTP client state storage is created (Expiration date/time=creation time of PTPCLTABLE record plus configured maximum lifetime). The PTP server SHOULD erase entries from this table after the expiration time of the key pair has been reached.

The PTP server, after receiving a signaling message, will perform the following steps:

- \* if the request contains an NTS\_TLV with ntsMsgId == NTS\_INIT, it decrypts the cookie with its master key K to obtain the two C2S and S2C keys, for all other ntsMsgId values it MUST perform a lookup into the PTPCLTABLE
- \* if the request contains an NTS\_TLV with ntsMsgId == NTS\_REQUEST, the nonce used in this signaling message is identical to the NEXT\_NONCE stored for this client in the PTPCLTABLE
- \* if the request contains an NTS\_TLV with ntsMsgId == NTS\_CHALLENGE\_REQUEST, CHALLENGE\_EXP has not been reached and the cookie used in this signaling message is identical to the CHALLENGE\_NONCE stored for this client
- \* If either the cookie cannot be decrypted, no matching entry could be found in the PTPCLTABLE, the nonce does not match the NEXT\_NONCE or the CHALLENGE\_NONCE, the message MUST be ignored. In all other cases, the following additional checks MUST be performed:
  - the ICV in the AUTHENTICATION\_TLV is correct (using the C2S key from the provided cookie or from the matching entry in the PTPCLTABLE)
  - the key expiration date/time has not been reached

If the CHALLENGE\_NONCE check fails (only applicable when the ntsMsgId in the received message is NTS\_CHALLENGE\_RESPONSE), the message MUST be ignored.

If the key expiration check fails, the request is denied and, by setting the key lifetime field of the NTS\_TLV in the response to 0 and the ntsMsgId to NTS\_INVALIDKEYS, the client is told to obtain new keys and a new cookie from a NTS-KE server before it can establish a new contract with this PTP server.

If no matching entry exists in the PTPCLTABLE, or if the checks above result require an NTS\_CHALLENGE\_REQUEST response, any active contract (if there is one) MUST NOT be changed, canceled or otherwise modified. This is to avoid that an attacker sends an invalid request which stops the currently active contract and therefore successfully carries out a denial-of-service attack.

When a PTP server receives a NTS\_INIT message with a valid cookie, the following challenge/response procedure MUST be followed:

- \* The PTP server will deny the REQUEST\_UNICAST\_TRANSMISSION\_TLV request and responds with an NTS\_MessageId NTS\_CHALLENGE\_REQUEST and a nonce which it stores as the CHALLENGE\_NONCE in the PTPCLTABLE. The server will put the sequenceId from the PTP packet header of the request into the reqSeqId field of the NTS\_TLV
- \* The client, after receiving the response with the NTS\_CHALLENGE\_REQUEST message ID SHALL check that the reqSeqId is identical to the sequenceId of the request it sent to the server. If this fails, the message MUST be ignored by the client.
- \* If the reqSeqId check is successful, the client resends the REQUEST\_UNICAST\_TRANSMISSION\_TLV using a ntsMsgId of NTS\_CHALLENGE\_RESPONSE and MUST use the cookie it received with the NTS\_CHALLENGE\_REQUEST response.
- \* The server then checks that the cookie presented by the client in the NTS\_CHALLENGE\_RESPONSE response is identical to the CHALLENGE\_NONCE. If that is true, the client can be trusted and the REQUEST\_UNICAST\_TRANSMISSION\_TLV included in this response is considered to be a valid and trusted request.

After the successful verification of the request, the REQUEST\_UNICAST\_TRANSMISSION\_TLV is processed according to [IEEE1588].

In case the PTP server grants the request to the client, the process moves on to the 3rd phase, the packet transmission phase. If the PTP server denies the request for whatever reason (i.e. it has no capacity or its state does not allow to reliably transmit the packets at the requested rate), it sends a unicast grant denial message, i.e. a PTP signaling message carrying a GRANT\_UNICAST\_TRANSMISSION\_TLV with the grant duration set to zero.

If a PTP client receives a GRANT\_UNICAST\_TRANSMISSION\_TLV containing an NTS\_TLV with a correct ICV and a key lifetime set to 0, it MUST delete all cookies it holds for that PTP server as well as the S2C/C2S key pair and cease all communication until it re-ran phase 1 and obtained a new key pair and a new cookie.

Using the "maximum key lifetime" configuration parameter, a PTP server operator can prioritize memory requirements and required network traffic volume. A small maximum lifetime value results in PTPCLTABLE entries being deleted earlier, requiring more NTS\_CHALLENGE\_REQUEST exchanges between PTP client and PTP server. A large value may result in requiring fewer such packet exchanges but increases the memory consumption because PTPCLTABLE entries have to be stored for a longer period of time.

### 3.3. Phase 3: PTP Unicast Packet Transmission Phase

When the transmission request has been granted by the PTP server for a specific messagetype/duration, for all messagetypes except delay responses the GM immediately starts transmitting the messages in the requested rate. DELAY\_RESP messages will be sent after the client sent a DELAY\_REQ message.

All unicast PTP messages sent by the PTP server to the PTP client due to an active contract SHALL be secured by an AUTHENTICATION\_TLV that carries an ICV as described in [IEEE1588], subclause 16.14 (PTP Integrated Security). The ICV is created using the MAC function of the AEAD algorithm and the S2C key established between the PTP client and the NTS-KE server during the NTS-KE phase. All messages sent by the PTP client to the PTP server will be secured with the same mechanism, but using the C2S key.

All PTP messages in the packet transmission phase that do not carry an AUTHENTICATION\_TLV or in which the ICV is not correct MUST be ignored by the recipient.

In order to establish a protection against replay attacks in this phase, both the PTP client and the PTP server MUST check that the sequenceId of an incoming message is larger than the sequenceId of the same PTP message type received in the previous message. If an incoming message does not pass the sequenceId check, it MUST be ignored. Both PTP client and PTP server SHOULD allow to configure the maximum difference between the sequenceId values of two consecutive messages of the same message type. They MUST gracefully handle the rollover of a sequenceId, which is a unsigned int16 value (0-65535).

To avoid that an attacker resends a PTP message with a sequenceId that has been obtained before the last rollover, additional integrity checks SHOULD be applied. The maximum packet rate is 128 packets/second. Therefore, for each PTP message type sent at the maximum rate, the sequenceId rollover happens every 512 seconds (8 minutes, 32 seconds) as a minimum. For SYNC, DELAY\_REQ and ANNOUNCE messages the recipient SHOULD check that the originTimestamp in the packet

does not differ more than 8 minutes from the `originTimestamp` of the previously received message. For `DELAY_RESP` messages, the `receiveTimestamp` SHOULD be used instead.

The packet transmission phase ends either when the contract expired or when either the PTP server or the PTP client cancels the contract.

#### 4. New NTS record types

##### 4.1. Cookie for Unicast PTP

The content of this NTS record is identical to record type 5 as defined in 4.1.6 of [RFC8915]. However, a NTS-KE server MUST send exactly one record of this type when PTP is negotiated as a next protocol.

##### 4.2. Unicast PTP Server Negotiation

The PTP server Negotiation record has a Record Type number of 8. Its body consists of an ASCII-encoded [RFC0020] string. The contents of the string SHALL be either an IPv4 address, an IPv6 address, or a fully qualified domain name (FQDN). IPv4 addresses MUST be in dotted decimal notation. IPv6 addresses MUST conform to the "Text Representation of Addresses" as specified in RFC 4291 [RFC4291] and MUST NOT include [RFC6874]. If a label contains at least one non-ASCII character, it is an internationalized domain name, and an A-LABEL MUST be used as defined in Section 2.3.2.1 of RFC 5890 [RFC5890]. If the record contains a domain name, the recipient MUST treat it as a FQDN, e.g., by making sure it ends with a dot.

When PTP is negotiated as a Next Protocol and this record is sent by the server, the body specifies the hostname or IP address of the PTP unicast server with which the client MUST associate and that will accept the supplied cookies. If no record of this type is sent, the client SHALL interpret this as a directive to associate with a PTP server at the same IP address as the NTS-KE server. Servers MUST NOT send more than one record of this type. If the record contains a FQDN which resolves to multiple addresses, the client MUST choose at least one of the addresses the FQDN resolves to. The client MAY choose to use more than one address to request synchronization services from multiple unicast PTP servers in parallel.

When this record is sent by the client, it indicates that the client wishes to associate with the specified PTP server. The NTS-KE server MAY incorporate this request when deciding which PTP server Negotiation records to respond with, but honoring the client's preference is OPTIONAL. The client MUST NOT send more than one record of this type.

If the client has sent a record of this type, the NTS-KE server SHOULD reply with the same record if it is valid and the server is able to supply cookies for it. If the client has not sent any record of this type, the NTS-KE server SHOULD respond with either a PTP server address in the same family as the NTS-KE session or a FQDN that can be resolved to an address in that family, if such alternatives are available.

Servers MAY set the Critical Bit on records of this type; clients SHOULD NOT.

#### 5. The PTP client Table

The PTP server MUST store the following data for each PTP client in a NTS PTP client Table (PTPCLTABLE):

- \* the S2C/C2S key pair
- \* the time when the validity of this key pair expires (KEY\_PAIR\_EXP)
- \* the next nonce to be expected from the client (NEXT\_NONCE)
- \* the nonce to be expected from the client in a NTS\_CHALLENGE\_RESPONSE message (CHALLENGE\_NONCE)
- \* the time when the validity of the CHALLENGE\_NONCE expires (CHALLENGE\_EXP)

#### 6. The NTS\_TLV

The NTS\_TLV contains:

1. tlvType: ORGANIZATION\_EXTENSION\_DO\_NOT\_PROPAGATE (8000 hex)
2. lengthField: number of octets in the value + 6
3. organizationID: IETF (=00005E hex)
4. organizationSubtype: TBD (needs to be assigned)
5. ntsMsgId: NTS Message Id (see below)
6. networkProtocol: Transport type (0x01=IPv4, 0x02=IPv6, 0x03=Ethernet), unsigned int
7. tSrcAddr: Transport source address of the sender, e.g. the IP address or Ethernet MAC address, 16 octets

8. keyLifetime: Key Lifetime in seconds, unsigned int32
9. reqSeqId: sequenceId of the request, unsigned int16
10. nonce/cookie: a nonce or, if ntsMsgId == NTS\_INIT, an NTS Cookie

Bits								Octets	TLV Offset
0	1	2	3	4	5	6	7		
tlvType								2	0
lengthField								2	2
organizationId								3	4
organizationSubType								3	7
ntsMsgId								1	10
networkProtocol								1	11
tSrcAddr								16	12
keyLifetime								4	28
reqSeqId								2	32
nonce/cookie								n	34

Figure 2: NTS TLV Format

The networkProtocol field allows to detect which transport protocol is in use and therefore how to interpret the tSrcAddr field. For an Ethernet MAC address, the 6 first octets of the tSrcAddr field are relevant, for an IPv4 address the first 4 octets are relevant and for an IPv6 address the full 16 octets are relevant. The enumeration is corresponding to the networkProtocol field as defined in [IEEE1588].

Please note that the size of the cookie depends on the chosen AEAD, it can therefore differ and has been placed at the end of the TLV. The lengthField allows a receiptient to calculate the length of the cookie.

The ntsMsgId field MUST contain one of the following values:

- \* 0x01 NTS\_INIT (for the first unicast negotiation message sent by the PTP client to the PTP server after completing a NTS-KE Phase in which the PTP client obtained a new key pair)
- \* 0x02 NTS\_REQUEST (for unicast negotiation messages sent by the PTP client to the PTP server)
- \* 0x03 NTS\_RESPONSE (for unicast negotiation messages sent by the PTP server to the PTP client)
- \* 0x04 NTS\_CHALLENGE\_REQUEST (for messages sent by the PTP server to the PTP client in case of a challenge/response procedure)
- \* 0x05 NTS\_CHALLENGE\_RESPONSE (for messages sent by the PTP client to the PTP server as a response to a NTS\_CHALLENGE\_REQUEST)
- \* 0x06 NTS\_INVALIDKEYS (for messages sent by the PTP server to the PTP client when the S2C/C2S key pair expired or whenever the PTP server wants to force the PTP client to re-run the NTS-KE Phase)

The reqSeqId field MUST be set to 0 for all messages except those with a NTS\_CHALLENGE\_REQUEST message Id. In that specific case the field MUST contain the sequenceId of the message to which this NTS\_CHALLENGE\_REQUEST is a response.

When sending a REQUEST\_UNICAST\_TRANSMISSION\_TLV, the PTP client will add the NTS\_TLV containing a cookie for this PTP server. The key lifetime SHALL always be set to 0 for all communication from the PTP client to the PTP server.

When sending a GRANT\_UNICAST\_TRANSMISSION\_TLV, the PTP server will add the NTS\_TLV as well. If the request of the client is granted (duration > 0), the NTS\_TLV will contain a new cookie and the key lifetime field SHALL represent the number of seconds the C2S and S2C keys continue to be valid.

The PTP server SHALL set a key lifetime of 0 when the lifetime of the S2C/C2S key pair expired. This allows the PTP server to force the client to re-start and obtain fresh keys and cookies from the NTS-KE server. When sending an NTS\_TLV with the key lifetime set to 0, the PTP server SHALL use the S2C key of the expired key pair to form the ICV in the AUTHENTICATION\_TLV, allowing the client to verify that the message has been sent by the PTP server.



When a client receives any message with a valid ICV but a key lifetime of 0 in the NTS\_TLV, it SHALL delete all cookies cease to send any messages to the PTP server and only restores communication with it after it obtained a new S2C/C2S key pair and a new set of cookies from the NTS-KE server.

## 7. IANA Considerations

RFC EDITOR: A new entry for Unicast PTP is required in the IANA Network Time Security Next Protocols Registry. The authors propose to add an entry with the Protocol ID = 1, the Protocol Name = "Unicast Precision Time Protocol" and a Reference to this draft. Please remove this comment before publishing and replace it with the data for the newly created entry from IANA.

## 8. Security Considerations

### 8.1. Threat Model

The procedures and mechanisms described in this draft protect against the following scenarios:

- \* Man-in-the-Middle (MITM) attack: All PTP nodes can verify that PTP messages received from another PTP server have not been modified in transit by an attacker. This is achieved by verifying that the ICV in the AUTHENTICATION\_TLV of every PTP message received is valid and has been created using the MAC function of the AEAD algorithm and the C2S or S2C key as established during the NTS-KE phase.
- \* Phase 2 Replay Attack (resending unmodified PTP unicast negotiation messages): PTP message integrity can be secured with the AUTHENTICATION\_TLV. However, with PTP this mechanism does not protect the transport protocol header and could therefore be used by an attacker to intercept a PTP message and then resending it using the same or a different source address. Or it could be resend at a later time to repeat a request or cancel an active contract. Resending it with the same address can be used to try and establish a contract for a unicast PTP client that is no longer requiring the service, requires the service in a different form (e.g. different message rates) or it can be used to cancel a contract (when resending a CANCEL\_UNICAST\_TRANSMISSION\_TLV after the client established a new contract). This can interrupt a required service for a PTP client or result in the PTP server unnecessarily consuming resources. By storing the nonce that has been provided by the server and that MUST be used by the client in its next request (NEXT\_NONCE in PTPCLTABLE), an unmodified resent REQUEST\_UNICAST\_TRANSMISSION\_TLV will not be granted, despite the

fact that the ICV in the AUTHENTICATION\_TLV is valid. To provide a robust defense against replaying NTS\_INIT messages, the challenge/response mechanism (NTS\_CHALLENGE\_REQUEST/NTS\_CHALLENGE\_RESPONSE) will require a PTP client to apply the correct C2S key and therefore protects against replaying a previously sent valid message with a cookie that has been encrypted with a still valid master key K.

- \* Phase 3 Replay Attack (resending unmodified PTP unicast messages): In Phase 3 the PTP server is sending PTP SYNC and ANNOUNCE messages to the PTP client and the PTP client is sending DELAY\_REQ messages to the PTP server, which responds with DELAY\_RESP messages. An attacker could resend any of these packets. For SYNC messages, that would result in the PTP client receiving "old time", i.e. the timestamps in such a message would be outdated and could disrupt time synchronization of the PTP client. A resent ANNOUNCE message could carry outdated information and therefore could force the PTP client to drop this server as a valid time source or distrust the protocol in other ways. Resending DELAY\_REQ messages could consume resources on the PTP server and, if the server would send DELAY\_RESP message, on the PTP client as well. The client could also be negatively affected by resent DELAY\_RESP messages that carry outdated time. A valid protection against such an attack is checking the sequenceId of each incoming message and by applying additional integrity checks to messages passing the sequenceId checks. See Section 3.3 for a detailed description of how this can be achieved.
- \* Amplification Attack/Distributed Denial of Service: Resending a REQUEST\_UNICAST\_TRANSMISSION\_TLV with a different source IP address could result in the PTP server sending PTP messages to IP addresses that do not expect and require receiving these messages. Due to the nature of unicast PTP, the traffic amplification potential is very high because one PTP signaling message containing a REQUEST\_UNICAST\_TRANSMISSION\_TLV can generate thousands of PTP messages from the PTP server to the source IP address used in the request message. This is mitigated by including the IP address of the originator of a message as a field (tSrcAddr) in the NTS\_TLV. The TLV as part of the PTP message is protected by the ICV in the AUTHENTICATION\_TLV and therefore a modified source address can be detected.

## 8.2. General Security Features

In addition to the above outlined protection mechanisms against specific attack scenarios, this draft also includes a generic security feature:

- \* **Key Freshness:** The expiration of C2S/S2C key pairs requires clients to obtain a new key pair in a configurable interval, which limits the time an attacker has to break those keys.

## 9. Delay Attacks

If an attacker gains control over a part of the network infrastructure on the path between clients and server, it could be possible to delay the forwarding of unicast PTP messages without modifying them. Applying such a delay only in one direction (e.g. for SYNC packets sent from the server to the client) would create an asymmetry in the delay calculation and as a result the error in the delay calculation would cause a timing error on the client. This kind of attack cannot be mitigated by NTS4UPTP as the cryptographic protection only allows to ensure the integrity of messages, which is not corrupted by a delay attack.

In addition to securing the network infrastructure (i.e. routers and switches) against this threat, another possible mitigation strategy for the client is to check the calculated delay against a static limit (which could be configurable by the user or is defined by the requirements of the application) or a dynamic limit, which the client could determine periodically for example by applying suitable statistical methods to determine a change in the calculated delay that indicates that the potential time error would exceed the sync requirements of the application.

## 10. Acknowledgements

The authors would like to thank the following contributors for their valuable feedback:

- \* Martin Langer, MSc and Prof. Dr.-Ing. Rainer Bermbach, Ostfalia University

## 11. References

### 11.1. Normative References

- [RFC0020] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <<https://www.rfc-editor.org/info/rfc20>>.
- [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>>.

- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <<https://www.rfc-editor.org/info/rfc4291>>.
- [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>>.
- [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>>.
- [RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <<https://www.rfc-editor.org/info/rfc5890>>.
- [RFC6874] Carpenter, B., Cheshire, S., and R. Hinden, "Representing IPv6 Zone Identifiers in Address Literals and Uniform Resource Identifiers", RFC 6874, DOI 10.17487/RFC6874, February 2013, <<https://www.rfc-editor.org/info/rfc6874>>.

## 11.2. Informative References

- [IEEE1588] IEEE, "IEEE Std 1588-2019 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control", IEEE Standard 1588, 2019, <<https://standards.ieee.org/content/ieee-standards/en/standard/1588-2019.html>>.

## Authors' Addresses

Heiko Gerstung  
Meinberg Funkuhren GmbH&Co.KG  
Lange Wand 9  
31812 Bad Pyrmont  
Germany

Phone: +49 5281 9309 0  
Email: [heiko.gerstung@meinberg.de](mailto:heiko.gerstung@meinberg.de)  
URI: <http://www.meinbergglobal.com/>

Marius Rohde  
Meinberg Funkuhren GmbH&Co.KG  
Lange Wand 9

31812 Bad Pyrmont  
Germany

Phone: +49 5281 9309 0  
Email: [marius.rohde@meinberg.de](mailto:marius.rohde@meinberg.de)  
URI: <http://www.meinbergglobal.com/>

Douglas Arnold  
Meinberg USA Inc.  
100 Stony Point Road Suite 110  
Santa Rosa, CA 95401  
United States of America

Phone: +1 877-PTP-1588  
Email: [doug.arnold@meinberg-usa.com](mailto:doug.arnold@meinberg-usa.com)  
URI: <http://www.meinbergglobal.com/>

Network Time Protocol  
Internet-Draft  
Intended status: Informational  
Expires: 17 May 2022

J. Guessing  
Nederlandse Publieke Omroep  
13 November 2021

NTPv5 use cases and requirements  
draft-guessing-ntp-ntp5-requirements-04

## Abstract

This document describes the use cases, requirements, and considerations that should be factored in the design of a successor protocol to supersede version 4 of the NTP protocol [RFC5905] presently referred to as NTP version 5 ("NTPv5"). This document is non-exhaustive and does not in its current version represent working group consensus.

## Note to Readers

\_RFC Editor: please remove this section before publication\_

Source code and issues for this draft can be found at <https://github.com/fiestajetsam/draft-guessing-ntp-ntp5-requirements> (<https://github.com/fiestajetsam/draft-guessing-ntp-ntp5-requirements>).

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 17 May 2022.

## Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
1.1. Notational Conventions . . . . .	3
2. Use cases and existing deployments of NTP . . . . .	3
3. Requirements . . . . .	3
3.1. Resource management . . . . .	3
3.2. Algorithms . . . . .	4
3.3. Timescales . . . . .	4
3.4. Leap seconds . . . . .	5
3.5. Backwards compatibility to NTS and NTPv4 . . . . .	5
3.5.1. Dependent Specifications . . . . .	5
3.6. Extensibility . . . . .	5
3.7. Security . . . . .	6
4. Non-requirements . . . . .	6
4.1. Server malfeasance detection . . . . .	6
5. Threat model . . . . .	6
5.1. Delay-based attacks . . . . .	6
5.2. Payload manipulation . . . . .	7
5.3. Denial of Service and Amplification . . . . .	7
6. IANA Considerations . . . . .	7
7. Security Considerations . . . . .	7
8. References . . . . .	7
8.1. Normative References . . . . .	7
8.2. Informative References . . . . .	8
Appendix A. Acknowledgements . . . . .	8
Author's Address . . . . .	8

## 1. Introduction

NTP version 4 [RFC5905] has seen active use for over a decade, and within this time period the protocol has not only been extended to support new requirements but also fallen victim to vulnerabilities that have made it used for distributed denial of service (DDoS) amplification attacks.

### 1.1. Notational Conventions

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. Use cases and existing deployments of NTP

There are several common scenarios for existing NTPv4 deployments; publicly accessible NTP services such as the NTP Pool [ntppool] are used to offer clock synchronisation for end users and embedded devices, ISP provided servers to synchronise devices such as customer-premises equipment where reduced accuracy may be tolerable. Depending on the network and path these deployments may be affected by variable latency as well as throttling or blocking by providers.

Data centres and cloud computing providers also have deployed and offer NTP services both for internal use and for customers, particularly where the network is unable to offer or does not require PTP [IEEE-1588-2008]. As these deployments are less likely to be constrained by network latency or power the potential for higher levels of accuracy and precision within the bounds of the protocol are possible.

## 3. Requirements

At a high level, NTPv5 should be a protocol that is capable of operating in both local networks and also over public internet connections where packet loss, delay, and even filtering may occur. It should be able to provide enough information for both basic time information as well as synchronisation.

### 3.1. Resource management

Historically there have been many documented instances of NTP servers taking a large increase in unauthorised traffic [ntp-misuse] and the design of NTPv5 must ensure the risk of these can be minimised to the fullest extent.

Servers SHOULD have a new identifier that peers use as reference, this SHOULD NOT be a FQDN, an IP address or identifier tied to a public certificate. Servers SHOULD be able to migrate and change their identifiers as stratum topologies or network configuration changes occur.



The protocol **MUST** have the capability for servers to notify clients that the service is unavailable, and clients **MUST** have clearly defined behaviours honouring this signalling. In addition servers **SHOULD** be able to communicate to clients that they should reduce their query interval rate when the server is under high bandwidth or has reduced capacity.

Clients **SHOULD** re-establish connections with servers at an interval to prevent attempting to maintain connectivity to a dead host and give network operators the ability to move traffic away from hosts in a timely manner.

The protocol **SHOULD** have provisions for deployments where Network Address Translation occurs, and define behaviours when NAT rebinding occurs. This should also not compromise any DDoS mitigation(s) that the protocol may define.

### 3.2. Algorithms

The use of algorithms describing functions such as clock filtering, selection and clustering **SHOULD** have agility, allowing for implementations to develop and deploy new algorithms independantly. Signalling of algorithm use or preference **SHOULD NOT** be transmitted by servers.

The working group should consider creating a separate informational document to describe an algorithm to assist with implementation, and to consider adopting future documents which describe new algorithms as they are developed. Specifying client algorithms separately from the protocol allows will allow NTPv5 to meet the needs of applications with a variety of network properties and performance requirements.

### 3.3. Timescales

The protocol **SHOULD** adopt a linear, monotonic timescale as the basis for communicating time. The format should meet sufficient scale and precision with resolution either meeting or exceeding NTPv4, and have a rollover date sufficiently far enough into the future that the protocol's complete obsolescence is most likely to occur first.

The timescale in addition to any other time sensitive information **MUST** be sufficient to calculate representations of both UTC and TAI. Through extensions the protocol **SHOULD** support additional timescale representations outside of the main specification, and all transmissions of time data **SHALL** indicate the timescale in use.

### 3.4. Leap seconds

Transmission of UTC leap second information MUST be included in the protocol in order for clients to generate a UTC representation but must be transmitted as separate information to the timescale. The specification SHOULD also be capable of transmitting upcoming leap seconds greater than 1 calendar day in advance.

Leap second smearing SHOULD NOT be applied to timestamps transmitted by the server, however this should not prevent implementers from applying leap second smearing between the client and any clock it is training.

### 3.5. Backwards compatibility to NTS and NTPv4

The support for compatibility with other protocols should not prevent addressing issues that have previously caused issues in deployments or cause ossification of the protocol.

Protocol ossification MUST be addressed to prevent existing NTPv4 deployments which incorrectly respond to clients posing as NTPv5 from causing issues. Forward prevention of ossification (for a potential NTPv6 protocol in the future) should also be taken into consideration.

The model for backward compatibility is servers that support multiple versions NTP and send a response in the same version as the request. This does not preclude servers from acting as a client in one version of NTP and a server in another.

#### 3.5.1. Dependent Specifications

Many other documents make use of NTP's data formats ([RFC5905] Section 6) for representing time, notably for media and packet timestamp measurements. Any changes to the data formats should consider the potential implementation complexity that may be incurred.

### 3.6. Extensibility

The protocol MUST have the capability to be extended, and that implementations MUST ignore unknown extensions. Unknown extensions received by a server from a lower stratum server SHALL not be added to response messages sent by the server receiving these extensions.

### 3.7. Security

Data authentication and optional data confidentiality **MUST** be integrated into the protocol, and downgrade attacks by an in-path attacker must be mitigated.

Cryptographic agility must be available, allowing for the protocol to update to the use of more secure cryptographic primitives as they are developed and as attacks and vulnerabilities with incumbent primitives are discovered. Intermediate devices such as hardware capable of performing timestamping of packets **SHOULD** be able to include information to packets in flight without requiring modification or removal of authentication or confidentiality on the packet.

Consideration must be given around how this will be incorporated into any applicable trust model. Downgrading attacks that could lead to an adversary disabling or removing encryption or authentication **MUST NOT** be possible in the design of the protocol.

## 4. Non-requirements

This section covers topics that are explicitly out of scope.

### 4.1. Server malfeasance detection

Detection and reporting of server malfeasance should remain out of scope as [I-D.ietf-ntp-rough-time] already provides this capability as a core functionality of the protocol.

## 5. Threat model

The assumptions that apply to all of the threats and risks within this section are based on observations of the use cases defined earlier in this document, and focus on external threats outside of the trust boundaries which may be in place within a network. Internal threats and risks such as a trusted operator are out of scope.

### 5.1. Delay-based attacks

The risk that an on-path attacker can delay packets between a client and server exists in all time protocols operating on insecure networks and its mitigations are limited within the protocol with a clock which is not yet synchronised. Increased path diversity and protocol support for synchronisation across multiple heterogeneous sources are likely the most effective mitigations.

## 5.2. Payload manipulation

Conversely on-path attackers who can manipulate timestamps could also speed up a client's clock, also resulting into drift-related malfunctions and errors such as incorrect expiration of public certificates on the affected hosts. An attacker may also manipulate other data in flight to disrupt service and cause de-synchronisation. In both cases having message authentication with a regular key rotation interval should mitigate; however consideration should be made for hardware based timestamping.

## 5.3. Denial of Service and Amplification

NTPv4 has previously suffered from DDoS amplification attacks using a combination of IP address spoofing with a private mode commands used in many NTP implementations, leading to an attacker being able to orders of magnitude of traffic to a victim IP address. Current mitigation uses a combination of disabling the use of private mode commands, in addition to encouraging network operators to implement BCP 38 [RFC2827]. Additional mitigations in future protocol specification should reduce the amplification factor in request/response payload sizes [drdos-amplification] through the use of padding and consideration of payload data.

## 6. IANA Considerations

This document makes no requests of IANA.

## 7. Security Considerations

As this document is intended to create discussion and consensus, it introduces no security considerations of its own.

## 8. References

### 8.1. Normative References

- [I-D.ietf-ntp-rougtime] Malhotra, A., Langley, A., Ladd, W., and M. Dansarie, "Rougtime", Work in Progress, Internet-Draft, draft-ietf-ntp-rougtime-05, 24 May 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-ntp-rougtime-05>>.
- [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/rfc/rfc2119>>.

- [RFC2827] Ferguson, P. and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing", BCP 38, RFC 2827, DOI 10.17487/RFC2827, May 2000, <<https://www.rfc-editor.org/rfc/rfc2827>>.
- [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/rfc/rfc5905>>.
- [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/rfc/rfc8174>>.

## 8.2. Informative References

- [drdos-amplification] "Amplification and DRDoS Attack Defense -- A Survey and New Perspectives", n.d., <<https://arxiv.org/abs/1505.07892>>.
- [IEEE-1588-2008] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems", n.d..
- [ntp-misuse] "NTP server misuse and abuse", n.d., <[https://en.wikipedia.org/wiki/NTP\\_server\\_misuse\\_and\\_abuse](https://en.wikipedia.org/wiki/NTP_server_misuse_and_abuse)>.
- [ntppool] "pool.ntp.org: the internet cluster of ntp servers", n.d., <<https://www.ntppool.org>>.

## Appendix A. Acknowledgements

The author would like to thank Doug Arnold and Hal Murray for contributions to this document, and would like to acknowledge Daniel Franke, Watson Ladd, Miroslav Lichvar for their existing documents and ideas. The author would also like to thank Angelo Moriondo, Franz Karl Achard, and Malcom McLean for providing the author with motivation.

## Author's Address

James Guessing  
Nederlandse Publieke Omroep  
Postbus 26444

1202 JJ Hilversum  
Netherlands

Email: [james.ietf@gmail.com](mailto:james.ietf@gmail.com)

Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: 19 August 2022

M. Lichvar  
Red Hat  
15 February 2022

Network Time Protocol Version 5  
draft-mlichvar-ntp-ntp5-04

Abstract

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

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 19 August 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction . . . . .	2
1.1. Requirements Language . . . . .	3
2. Basic Concepts . . . . .	3
3. Data Types . . . . .	4
4. Message Format . . . . .	5
5. Extension Fields . . . . .	9
5.1. Padding Extension Field . . . . .	10
5.2. MAC Extension Field . . . . .	10
5.3. Reference IDs Request and Response Extension Fields . . .	10
5.4. Server Information Extension Field . . . . .	12
5.5. Correction Extension Field . . . . .	12
5.6. Reference Timestamp Extension Field . . . . .	15
5.7. Monotonic Timestamp Extension Field . . . . .	15
6. Measurement Modes . . . . .	16
7. Client Operation . . . . .	18
8. Server Operation . . . . .	20
9. NTPv5 Negotiation in NTPv4 . . . . .	22
10. Acknowledgements . . . . .	23
11. IANA Considerations . . . . .	23
12. Security Considerations . . . . .	23
13. References . . . . .	23
13.1. Normative References . . . . .	23
13.2. Informative References . . . . .	23
Author's Address . . . . .	24

## 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  $\log_2$  value of seconds. For example, a  $\log_2$  value of 4 is 2 to the power of 4 (16 seconds), or a  $\log_2$  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.

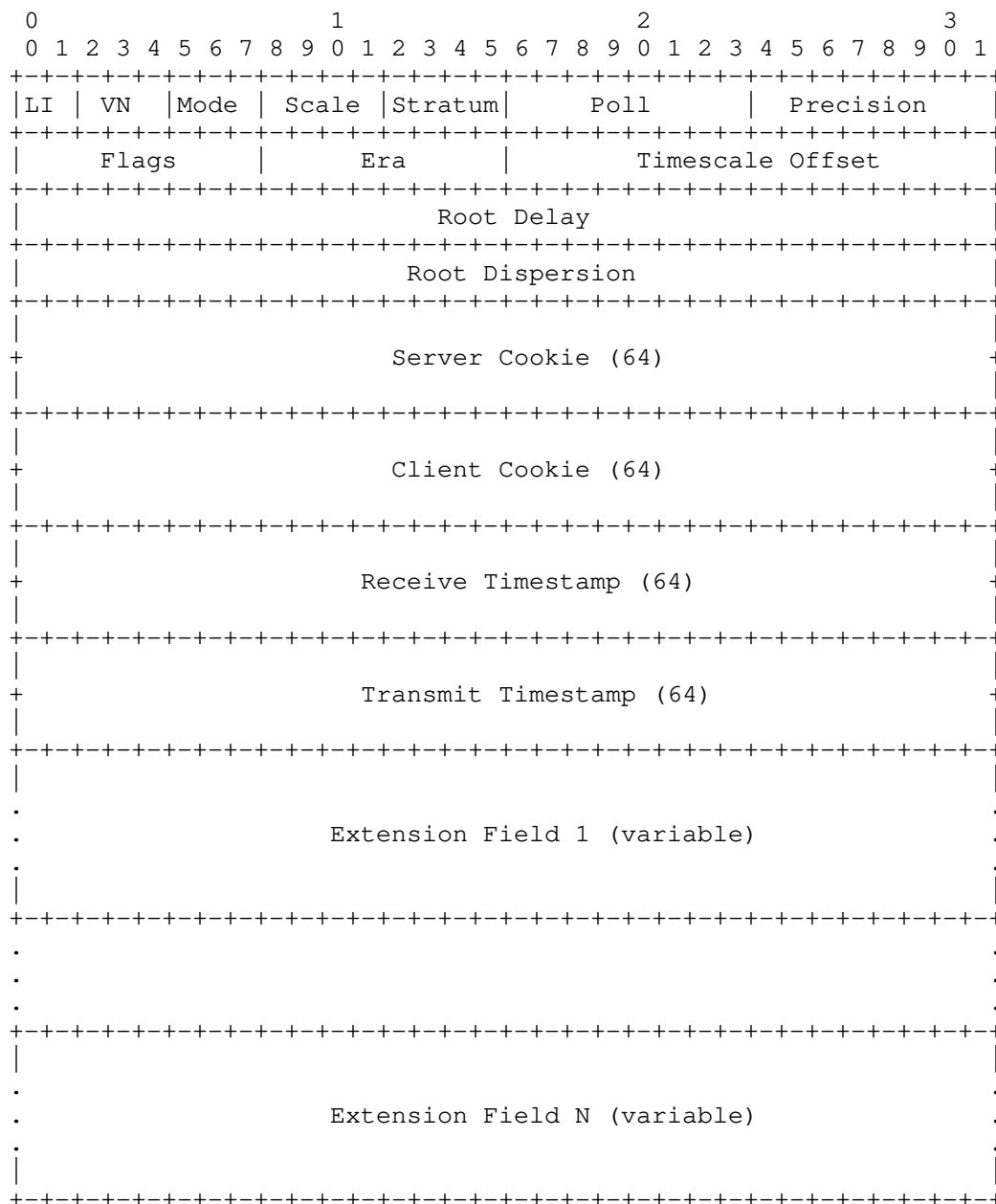


Figure 1: Format of NTPv5 messages

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 host. 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 zero. In responses it 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 values: synchronized (0) and not synchronized (3).

**0x4: Interleaved mode**

In requests it is a request for a response in the interleaved mode. In responses it 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 timel6 type (0x8000 if unknown).
- \* In the leap-smear UTC (3), it is the current offset between the leap smeared time and UTC (former minus latter) using the timel6 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.

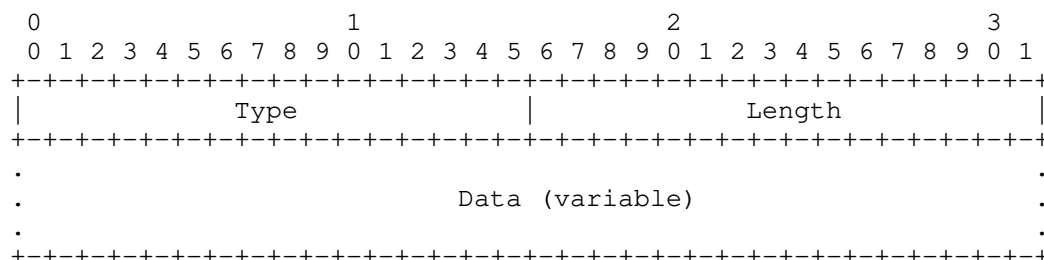


Figure 2: Format of NTPv5 extension fields

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 doesn't 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 doesn't 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 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. False positives are possible, but should be very rare for the specified length of the filter, even with a large number of reference IDs in the filter.

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.

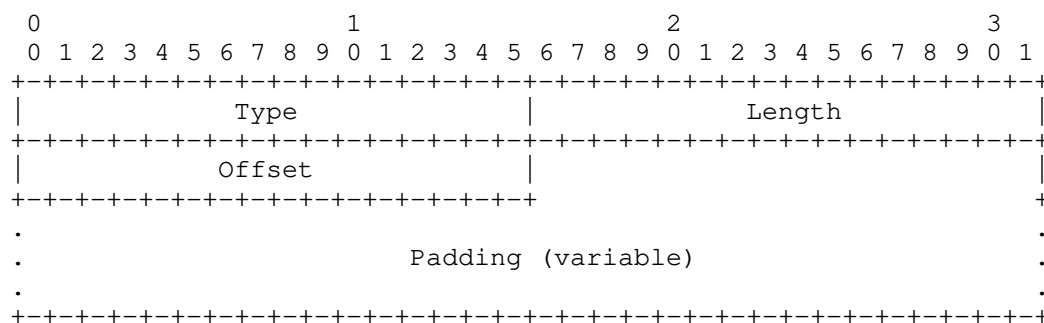


Figure 3: Format of Reference IDs Request Extension Field

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

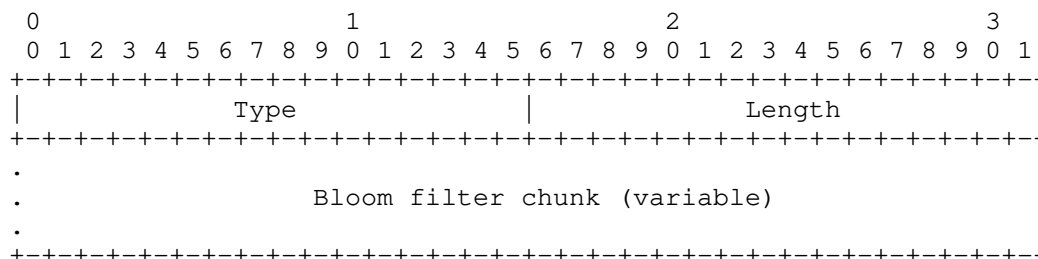


Figure 4: Format of Reference IDs Response Extension Field

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.

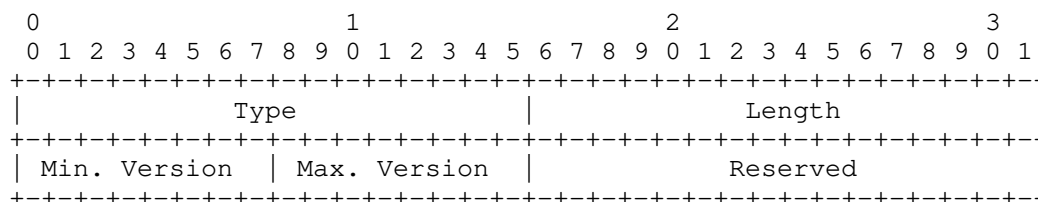


Figure 5: Format of Server Information Extension Field

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.

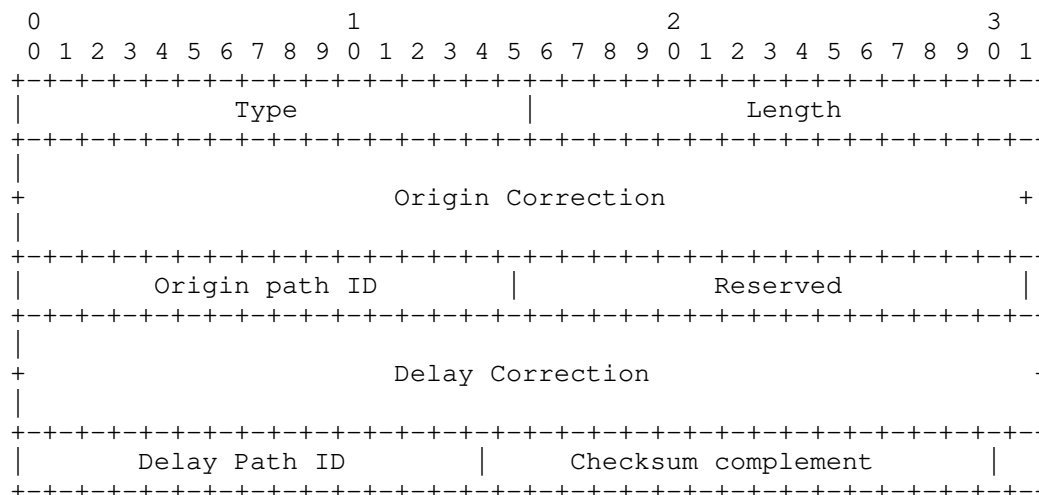


Figure 6: Format of Correction Extension Field

#### 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.)

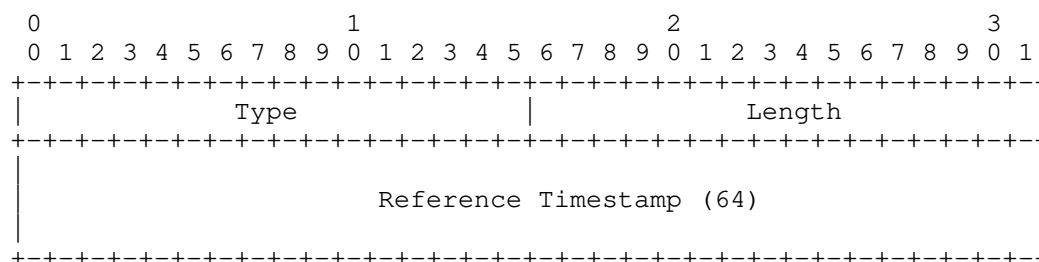


Figure 7: Format of Reference Timestamp Extension Field

#### 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 doesn't 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.

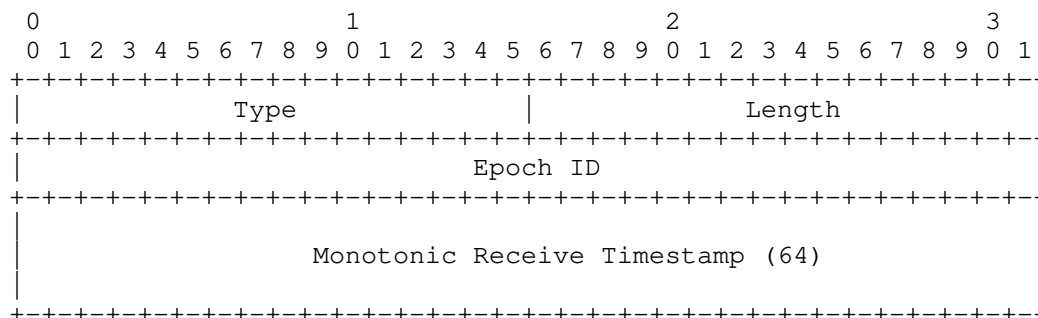


Figure 8: Format of Monotonic Timestamp Extension Field

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:

$$* \text{ offset} = ((T2 + T3) - (T4 + T1) + (Cd - Co)) / 2$$

$$* \text{ delay} = |(T4 - T1) - (T3 - T2) - (Cd + Co)|$$

$$* \text{ dispersion} = |T4 - T1| * DR$$

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 previous response was formed or sent.

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

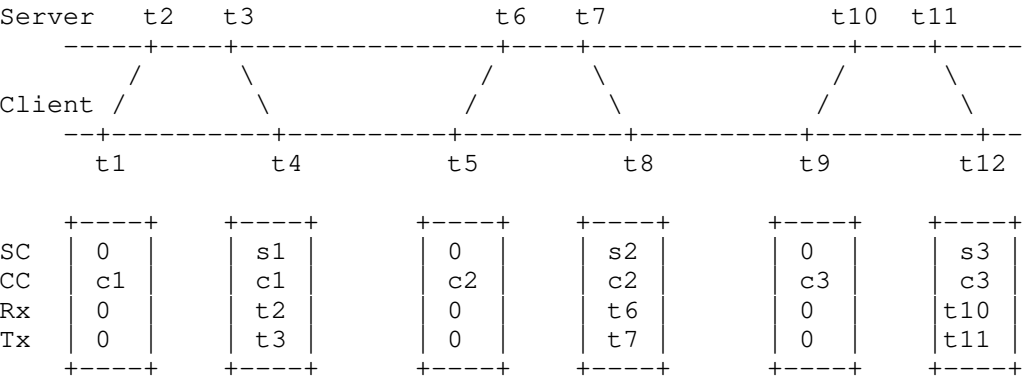


Figure 9: Cookies and timestamps in basic mode

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).

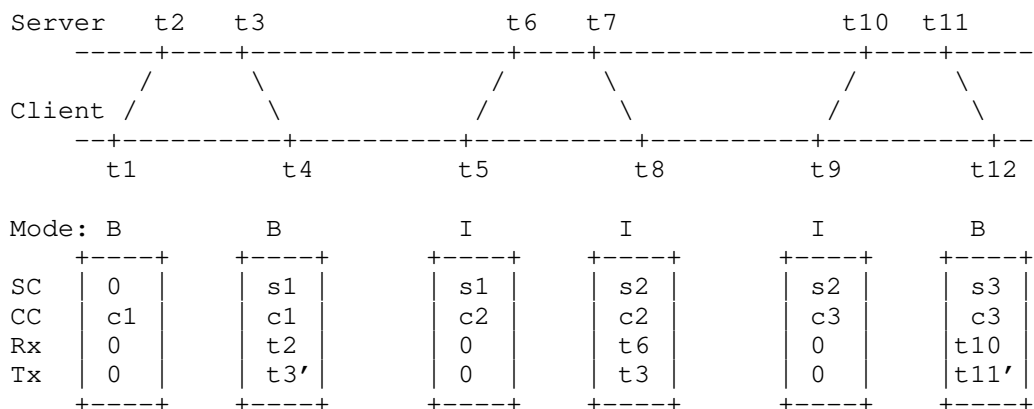


Figure 10: Cookies and timestamps in interleaved mode

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 a response in the interleaved mode.
  - \* Server cookie is set only in the interleaved mode. If a valid response from the server was received previously, it is set to the server cookie from the previous response.
  - \* 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 doesn't 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. The server MAY respond with an NTPv5 message if and only if the request has version 5.
3. Drops the request if the format is not valid, mode is not 3, or authentication fails if the MAC Extension Field or another authenticator field is present. 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 invalid 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 is 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. 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, and is not configured to use a particular version, SHOULD start with NTPv4 requests having the reference timestamp set to 0x4e5450354e545035. If the server responds with the same reference timestamp, the client SHOULD switch to NTPv5.

## 10. 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.

## 11. IANA Considerations

This memo includes no request to IANA.

## 12. Security Considerations

## 13. References

### 13.1. Normative References

- [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>>.

### 13.2. 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://www.ietf.org/archive/id/draft-ietf-ntp-interleaved-modes-07.txt>>.

[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>>.

#### Author's Address

Miroslav Lichvar  
Red Hat  
Purkynova 115  
612 00 Brno  
Czech Republic

Email: [mlichvar@redhat.com](mailto:mlichvar@redhat.com)

Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: 22 April 2022

M. Lichvar  
Red Hat  
19 October 2021

NTP Over PTP  
draft-mlichvar-ntp-over-ntp-01

## Abstract

This document specifies a transport for the Network Time Protocol (NTP) client-server mode using the Precision Time Protocol (PTP) to enable hardware timestamping on hardware that can timestamp PTP messages but not NTP messages.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 22 April 2022.

## Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. PTP transport for NTP . . . . .	3
3. Implementation Status - RFC EDITOR: REMOVE BEFORE PUBLICATION . . . . .	5
4. Security Considerations . . . . .	5
5. References . . . . .	5
5.1. Normative References . . . . .	6
5.2. Informative References . . . . .	6
Author's Address . . . . .	6

## 1. Introduction

The Precision Time Protocol (PTP) [IEEE1588] was designed for highly accurate synchronization of clocks in a network. It relies on hardware timestamping supported in network devices (e.g. interface controllers, switches, and routers) to eliminate the impact of processing and queueing delays on PTP measurements.

PTP was originally designed for multicast communication. Later was added a unicast mode, which can be used in larger networks with partial on-path PTP support (e.g. telecom profiles G.8265.1 and G.8275.2).

The Network Time Protocol [RFC5905] does not rely on hardware timestamping support, but implementations can use it if it is available to avoid the impact of processing and queueing delays, similarly to PTP. The client-server mode of NTP is functionally similar to the PTP unicast mode.

An issue for NTP is hardware that can specifically timestamp only PTP packets. This limitation comes from their design, which does not allow the timestamps to be captured or retrieved at the same rate as packets can be received or transmitted. A filter needs to be implemented in the hardware to inspect each packet and timestamp only those that actually need it. The filter can be usually configured for the PTP transport (e.g. UDPv4, UDPv6, 802.3) and sometimes even the message type (e.g. sync message or delay request) to further reduce the rate of timestamps on the server or client side. This limitation prevents hardware timestamping of NTP messages. It also prevents timestamping of PTP messages if they are secured at the transport layer or below (e.g. IPSec or MACSec).

This document specifies a new transport for NTP to enable the PTP-specific timestamping support. It adds a new extension field (TLV) for PTP to contain NTP messages.



NTP over PTP does not disrupt normal operation of PTP. A network and even a single host can support both at the same time.

The specification does not take advantage of the PTP correctionField modified by PTP transparent clocks as their support for the unicast mode seems to be rare or nonexistent.

The client/server mode of NTP, even if using the PTP transport, has several advantages when compared to the PTP unicast mode:

- \* It is more secure. It can use existing security mechanisms specified for NTP like Network Time Security [RFC8915], not losing any of its features. The PTP unicast mode allows an almost-infinite traffic amplification, which can be exploited for denial-of-service attacks and can only be limited by security mechanisms using client authentication.
- \* It needs fewer messages and less network bandwidth to get the same number of timestamps.
- \* It is better suited for synchronization in networks without full on-path support. It does not assume the network delay is constant and the number of measurements in opposite directions is symmetric (in PTP sync messages and delay requests have independent timing).

### 1.1. Requirements Language

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 [RFC2119].

## 2. PTP transport for NTP

A new TLV is defined for PTP to contain NTP messages in the client, server, and symmetric modes. Using other NTP modes in the TLV is not specified. Any transport specified for PTP that supports unicast messaging can be used for NTP over PTP, e.g. UDP on IPv4 and IPv6.

The type value of the NTP TLV is TBD. The TLV contains the whole NTP message as would normally be the UDP payload, without any modifications. The TLV does not propagate through boundary clocks.

If the UDP transport is used for PTP, the UDP source and destination port numbers MUST be the PTP event port (319). Client port randomization would break the timestamping.

The NTP TLV MUST be included in a delay request message. The originTimestamp field and all fields of the header SHOULD be zero, except:

- \* messageType is 1 (delay request)
- \* versionPTP is 2
- \* messageLength is the length of the PTP message including the NTP TLV
- \* domainNumber is 123
- \* flagField has the unicastFlag (0x4) bit set

An NTP client using the PTP transport sends NTP requests in PTP messages to the server at the same rate as it would normally send them over UDP.

A server which supports the NTP TLV MUST check for the domainNumber of 123 and respond to an NTP request with a single PTP message containing the NTP response using the same PTP message format. It MUST NOT send a delay response message.

A server which does not support the NTP TLV will not recognize the domain number and ignore the message. If it responded to messages in the domain (e.g. due to misconfiguration), it would send a delay response (to port 320 if using the UDP transport), which would be ignored by the client.

Any authenticator fields included in the NTP messages MUST be calculated only over the NTP message following the header of the NTP TLV.

Timestamps SHOULD NOT be adjusted for the beginning of the NTP data in the PTP message. They SHOULD still correspond to the ending of the transmission and beginning of the reception (e.g. start of delimiter in the Ethernet frame).

Any modifications of the correctionField made by potential one-step end-to-end transparent clocks in the network SHOULD be ignored by the server and client.

### 3. Implementation Status - RFC EDITOR: REMOVE BEFORE PUBLICATION

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in RFC 7942. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to RFC 7942, "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

#### 3.1. chrony

chrony (<https://chrony.tuxfamily.org>) has experimental support for PTP-over-NTP in its development branch. As the type of the NTP TLV, it uses 0x2023 from the experimental "do not propagate" range.

It was tested on Linux with the following network controllers, which have hardware timestamping limited to PTP packets:

Intel XL710 (i40e driver) - works

Intel X540-AT2 (ixgbe driver) - works

Intel 82576 (igb driver) - works

Broadcom BCM5720 (tg3 driver) - works

Broadcom BCM57810 (bnx2x driver) - does not timestamp unicast PTP packets

### 4. Security Considerations

The PTP transport prevents NTP clients from randomizing their source port. It has no other impact on security of NTP.

### 5. References

### 5.1. Normative References

- [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>>.
- [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>>.
- [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>>.

### 5.2. Informative References

- [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](mailto:mlichvar@redhat.com)