

INTERNET-DRAFT
Intended Status: Standards Track
Updates: 5905

V. Choudhary
A. Kumar
Huawei Technologies India
U. Windl
Universitaet Regensburg
June 3, 2015

Expires: December 5, 2015

The Network Time Protocol Version 4 Extension Format
draft-choudharykumar-ntp-ntpv4-extended-extensions-01

Abstract

The Network Time Protocol Version 4 (NTPv4) defines the optional usage of extension fields. An extension field as defined in RFC 5905, is an optional field that resides at the end of the NTP header and can be used to add optional capabilities or additional information that is not conveyed in the standard NTP header.

This document redefines NTP extension header proposed in RFC 5905 for addressing unnecessary padding issue which is required for differentiating between NTP extension and MAC data.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

Copyright and License Notice

Copyright (c) 2015 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 (<http://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
1.2	Terms & Abbreviations	3
2.	NTPv4 Extension Format	4
2.1	NTPv4 extension header format.	4
2.2.	Rules for new extensions.	4
2.3.	Ensuring backward compatibility.	5
3	Security Considerations	6
4	IANA Considerations	6
5	References	6
5.1	Normative References	6
5.2	Informative References	6
	Authors' Addresses	7

1 Introduction

In NTPv4, one or more extension fields can be inserted after the header and before the MAC, if a MAC is present. If a MAC is not present, one or more extension fields can be inserted after the header, according to the following rules:

- o If the packet includes a single extension field, the length of the extension field MUST be at least 7 words, i.e., at least 28 octets.
- o If the packet includes more than one extension field, the length of the last extension field MUST be at least 28 octets. The length of the other extension fields in this case MUST be at least 16 octets each.

This document updates RFC5905 by redefining legacy NTP extension header format which will help in reducing NTP packet size and ensures network bandwidth is not wasted just by adding unnecessary padding as mentioned above. All the new extensions MUST follow the proposed NTP extension header format.

1.1 Terminology

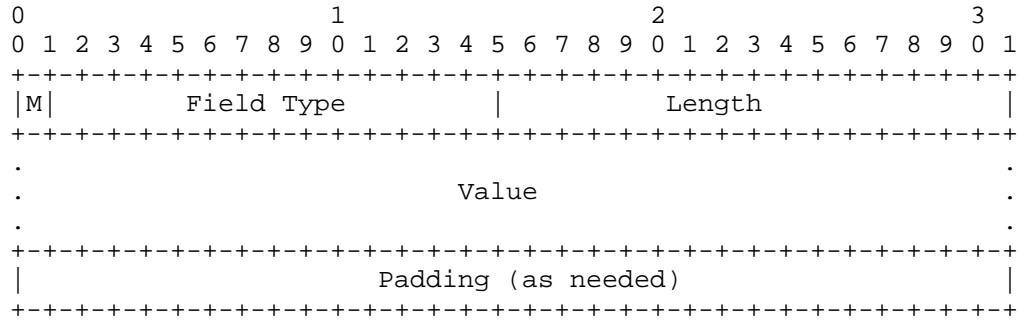
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [KEYWORDS].

1.2 Terms & Abbreviations

NTPv4	Network Time Protocol Version 4 [RFC5905]
MAC	Message Authentication Code

2. NTPv4 Extension Format

2.1 NTPv4 extension header format.



M-Bit : 1 - More extensions to be followed
 0 - No more extensions

- o. Removes size restriction for all extension fields
- o. Removal of padding saves network bandwidth
- o. Backward compatibility is ensured

Note: The proposed solution doesn't help in addressing legacy size constrains for NTP auto key extensions.

2.2. Rules for new extensions.

New NTPv4 extensions MUST follow below rules :

- o. Autokey extension/s MUST be continuous and SHOULD start after NTP header.

Proposed M-BIT solution cannot be used for Autokey extension as the MSB (most significant bit) of Autokey Field type is already used for indicating response messages. If the Autokey extensions are used in the middle then it won't be possible to indicate whether more/none extensions are following the current extension or not. Thus Autokey extensions MUST be the first extension after NTP header and SHOULD be processed as per the guideline defined by RFC 5906 [RFC5906].

- o. First extension following NTP header MUST be at least 28 octets.

This is required for handling legacy MAC issue.

- o. First extension following utokey extension MUST be at least 28 octets.

2.3. Ensuring backward compatibility.

- a. NTP packet with no extension/s and MAC.

This is simple as the packet size will be just equal to NTP header size. No special processing required.

- b. NTP packet with MAC only.

NTP packet size as defined in RFC 5905 will be considered for MAC processing.

- c. NTP packet with Autokey extension/s and with/without MAC.

Autokey extension's are the only defined NTP extension's so current size limitation and processing will remain as defined in RFC 5905 and 5906.

3 Security Considerations

The security considerations of the network time protocol are discussed in [RFC5905]. This document extends current available extension format for the usage of the NTP extension field, and thus the described in this document does not introduce new security considerations.

4 IANA Considerations

There are no new IANA considerations implied by this document. Field type value for new extensions should consider M-BIT while requesting IANA.

Example: If a new extension has field type value 0x0009 then it must request following values to be reserved by IANA.

0x0009 New extension field type (Without M-BIT)

0x8009 New extension field type (With M-BIT)

5 References

5.1 Normative References

- [KEYWORDS] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5905] Mills, D., Martin, J., Burbank, J., Kasch, W., "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

5.2 Informative References

- [RFC5906] Haberman, B., Mills, D., "Network Time Protocol Version 4: Autokey Specification", RFC 5906, June 2010.

Authors' Addresses

Vikram Choudhary
Huawei Technologies India Pvt. Ltd,
Divyashree Techno Park, Whitefield,
Bangalore, Karnataka 560037,
India

Email: vikram.choudhary@huawei.com

Anil Kumar S N
Huawei Technologies India Pvt. Ltd,
Divyashree Techno Park, Whitefield,
Bangalore, Karnataka 560037,
India

Email: anil.sn@huawei.com

Ulrich Windl
Universitaet Regensburg, Klinikum

EMail: ulrich.windl@rz.uni-regensburg.de

NTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 7, 2016

D. Sibold
K. Teichel
PTB
S. Roettger
Google Inc.
R. Housley
Vigil Security
July 06, 2015

Protecting Network Time Security Messages with the Cryptographic Message
Syntax (CMS)
draft-ietf-ntp-cms-for-nts-message-04

Abstract

This document describes a convention for using the Cryptographic Message Syntax (CMS) to protect the messages in the Network Time Security (NTS) protocol. NTS provides authentication of time servers as well as integrity protection of time synchronization messages using Network Time Protocol (NTP) or Precision Time Protocol (PTP).

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].

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 <http://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 January 7, 2016.

Copyright Notice

Copyright (c) 2015 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 (http://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. CMS Conventions for NTS Message Protection 3
 - 2.1. Fields of the employed CMS Content Types 5
 - 2.1.1. ContentInfo 5
 - 2.1.2. SignedData 6
 - 2.1.3. EnvelopedData 8
- 3. Implementation Notes: ASN.1 Structures and Use of the CMS . . 9
 - 3.1. Preliminaries 9
 - 3.2. Unicast Messages 9
 - 3.2.1. Association Messages 9
 - 3.2.2. Cookie Messages 10
 - 3.2.3. Time Synchronization Messages 11
 - 3.3. Broadcast Messages 12
 - 3.3.1. Broadcast Parameter Messages 12
 - 3.3.2. Broadcast Time Synchronization Message 13
 - 3.3.3. Broadcast Keycheck 14
- 4. Certificate Conventions 14
- 5. IANA Considerations 15
- 6. Security Considerations 15
- 7. References 15
 - 7.1. Normative References 15
 - 7.2. Informative References 15
- Appendix A. ASN.1 Module 16
- Authors' Addresses 16

1. Introduction

This document provides details on how to construct NTS messages in practice. NTS provides secure time synchronization with time servers using Network Time Protocol (NTP) [RFC5905] or Precision Time Protocol (PTP) [IEEE1588]. Among other things, this document

describes a convention for using the Cryptographic Message Syntax (CMS) [RFC5652] to protect messages in the Network Time Security (NTS) protocol. Encryption is used to provide confidentiality of secrets, and digital signatures are used to provide authentication and integrity of content.

Sometimes CMS is used in an exclusively ASN.1 [ASN1] environment. In this case, the NTS message may use any syntax that facilitates easy implementation.

2. CMS Conventions for NTS Message Protection

Regarding the usage of CMS, we differentiate between four archetypes according to which the NTS message types can be structured. They are presented below. Note that the NTS Message Object that is at the core of each structure does not necessarily contain all the data needed for the particular message type, but may contain only that data which needs to be secured directly with cryptographic operations using the CMS. Specific information about what is included can be found in Section 3.

NTS-Plain: This archetype is used for actual time synchronization messages (explicitly, the following message types: `time_request`, `time_response`, `server_broad`, see [I-D.ietf-ntp-network-time-security], Section 6) as well as for the very first messages of a unicast or a broadcast exchange (`client_assoc` or `client_bpar`, respectively) and the broadcast keycheck exchange (`client_keycheck` and `server_keycheck`). This archetype does not make use of any CMS structures at all. Figure 1 illustrates this structure.



NTS-Encrypted-and-Signed: This archetype is used for secure transmission of the cookie (only for the `server_cook` message type, see [I-D.ietf-ntp-network-time-security], Section 6). For this, the following CMS structure is used:

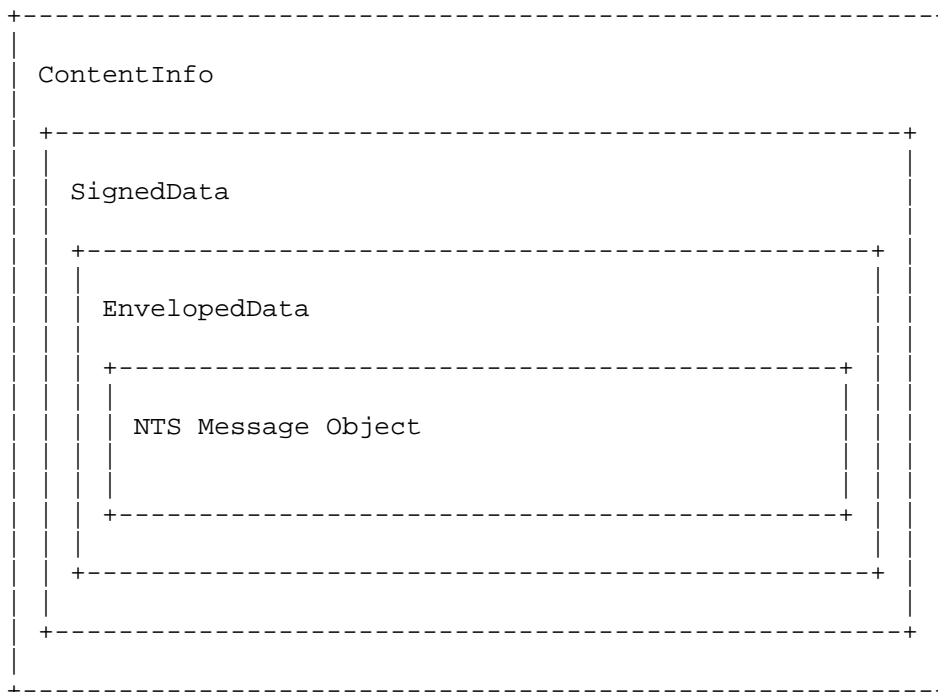
First, the NTS message MUST be encrypted using the `EnvelopedData` content type. `EnvelopedData` supports nearly any form of key management. In the NTS protocol the client provides a certificate in an unprotected message, and the

public key from this certificate, if it is valid, will be used to establish a pairwise symmetric key for the encryption of the protected NTS message.

Second, the EnvelopedData content MUST be digitally signed using the SignedData content type. SignedData supports nearly any form of digital signature, and in the NTS protocol the server will include its certificate within the SignedData content type.

Third, the SignedData content type MUST be encapsulated in a ContentInfo content type.

Figure 2 illustrates this structure.

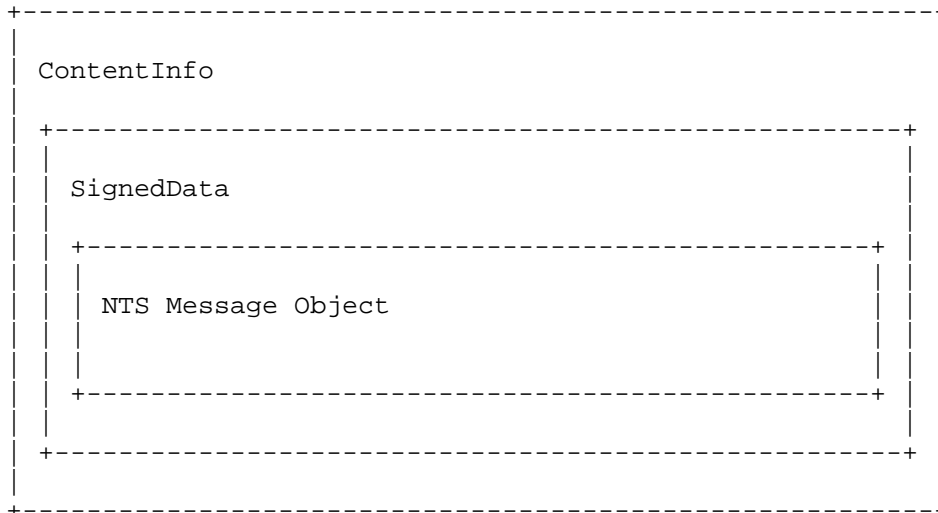


NTS-Signed: This archetype is used for server_assoc and server_bpar message types. It uses the following CMS structure:

First, the NTS message object MUST be wrapped in a SignedData content type. The messages MUST be digitally signed, and certificates included. SignedData supports nearly any form of digital signature, and in the NTS protocol the server will include its certificate within the SignedData content type.

Second, the SignedData content type MUST be encapsulated in a ContentInfo content type.

Figure 3 illustrates this structure.



NTS-Certified: This archetype is used for the client_cook message type. It uses a CMS structure much like the NTS-Signed archetype (see Figure 3), with the only difference being that messages SHOULD NOT be digitally signed. This archetype employs the CMS structure merely in order to transport certificates.

2.1. Fields of the employed CMS Content Types

Overall, three CMS content types are used for NTS messages by the archetypes above. Explicitly, those content types are ContentInfo, SignedData and EnvelopedData. The following is a description of how the fields of those content types are used in detail.

2.1.1. ContentInfo

The ContentInfo content type is used in all archetypes except NTS-Plain. The fields of the ContentInfo content type are used as follows:

contentType -- indicates the type of the associated content. For all archetypes which use ContentInfo (these are NTS-Certified, NTS-Signed and NTS-Encrypted-and-Signed), it MUST contain the object identifier for the SignedData content type:

```
id-signedData OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs7(7) 2 }
```

content -- is the associated content. For all archetypes using ContentInfo, it MUST contain the DER encoded SignedData content type.

2.1.2. SignedData

The SignedData content type is used in the NTS-Certified, NTS-Signed and NTS-Encrypted-and-Signed archetypes, but not in the NTS-Plain archetype. The fields of the SignedData content type are used as follows:

version -- the appropriate value depends on the optional items that are included. In the NTS protocol, the signer certificate MUST be included and other items MAY be included. The instructions in [RFC5652] Section 5.1 MUST be followed to set the correct value.

digestAlgorithms -- is a collection of message digest algorithm identifiers. In the NTS protocol, there MUST be exactly one algorithm identifier present. The instructions in Section 5.4 of [RFC5652] MUST be followed.

encapContentInfo -- this structure is always present. In the NTS protocol, it MUST follow these conventions:

eContentType -- is an object identifier. In the NTS protocol, for the NTS-Certified and NTS-Signed archetypes, it MUST identify the type of the NTS message that was encapsulated. For the NTS-Encrypted-and-Signed archetype, it MUST contain the object identifier for the EnvelopedData content type:

```
id-envelopedData OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs7(7) 3 }.
```

eContent is the content itself, carried as an octet string. For the NTS-Certified and NTS-Signed archetypes, it MUST contain the DER encoded encapsulated NTS message object. The instructions in Section 6.3 of [RFC5652] MUST be followed. For the NTS-Encrypted-and-Signed archetype, it MUST contain the DER encoded EnvelopedData content type.

certificates -- is a collection of certificates. In the NTS protocol, it MUST contain the DER encoded certificate [RFC5280] of the sender. It is intended that the collection of certificates be sufficient for the recipient to construct a certification path

from a recognized "root" or "top-level certification authority" to the certificate used by the sender.

`crls` -- is a collection of revocation status information. In the NTS protocol, it MAY contain one or more DER encoded CRLs [RFC5280]. It is intended that the collection contain information sufficient to determine whether the certificates in the `certificates` field are valid.

`signerInfos` -- is a collection of per-signer information. In the NTS protocol, for the NTS-Certified archetype, this SHOULD be left out. For both the NTS-Signed and the NTS-Encrypted-and-Signed archetypes, there MUST be exactly one `SignerInfo` structure present. The details of the `SignerInfo` type are discussed in Section 5.3 of [RFC5652]. In the NTS protocol, it MUST follow these conventions:

`version` -- is the syntax version number. In the NTS protocol, the `SignerIdentifier` is `subjectKeyIdentifier`, therefore the `version` MUST be 3.

`sid` -- identifies the signer's certificate. In the NTS protocol, the "sid" field contains the `subjectKeyIdentifier` from the signer's certificate.

`digestAlgorithm` -- identifies the message digest algorithm and any associated parameters used by the signer. In the NTS protocol, the identifier MUST match the single algorithm identifier present in the `digestAlgorithms`.

`signedAttrs` -- is a collection of attributes that are signed. In the NTS protocol, it MUST be present, and it MUST contain the following attributes:

`Content Type` -- see Section 11.1 of [RFC5652].

`Message Digest` -- see Section 11.2 of [RFC5652].

In addition, it MAY contain the following attributes:

`Signing Time` -- see Section 11.3 of [RFC5652].

`Binary Signing Time` -- see Section 3 of [RFC5652].

`signatureAlgorithm` -- identifies the signature algorithm and any associated parameters used by the signer to generate the digital signature.

signature is the result of digital signature generation using the message digest and the signer's private key. The instructions in Section 5.5 of [RFC5652] MUST be followed.

unsignedAttrs -- is an optional collection of attributes that are not signed. In the NTS protocol, it MUST be absent.

2.1.3. EnvelopedData

The EnvelopedData content type is used only in the NTS-Encrypted-and-Signed archetype. The fields of the EnvelopedData content type are used as follows:

version -- the appropriate value depends on the type of key management that is used. The instructions in [RFC5652] Section 6.1 MUST be followed to set the correct value.

originatorInfo -- this structure is present only if required by the key management algorithm. In the NTS protocol, it MUST be present when a key agreement algorithm is used, and it MUST be absent when a key transport algorithm is used. The instructions in Section 6.1 of [RFC5652] MUST be followed.

recipientInfos -- this structure is always present. In the NTS protocol, it MUST contain exactly one entry that allows the client to determine the key used to encrypt the NTS message. The instructions in Section 6.2 of [RFC5652] MUST be followed.

encryptedContentInfo -- this structure is always present. In the NTS protocol, it MUST follow these conventions:

contentType -- indicates the type of content. In the NTS protocol, it MUST identify the type of the NTS message that was encrypted.

contentEncryptionAlgorithm -- identifies the content-encryption algorithm and any associated parameters used to encrypt the content.

encryptedContent -- is the encrypted content. In the NTS protocol, it MUST contain the encrypted NTS message. The instructions in Section 6.3 of [RFC5652] MUST be followed.

unprotectedAttrs -- this structure is optional. In the NTS protocol, it MUST be absent.

3. Implementation Notes: ASN.1 Structures and Use of the CMS

This section presents some hints about the structures of the NTS message objects for the different message types when one wishes to implement the security mechanisms.

3.1. Preliminaries

The following ASN.1 coded data type "NTSNonce" is needed for other types used below for NTS messages. It specifies a 128 bit nonce as required in several message types:

```
NTSNonce ::= OCTET STRING (SIZE(16))
```

The following ASN.1 coded data types are also necessary for other types.

```
KeyEncryptionAlgorithmIdentifiers ::=
  SET OF KeyEncryptionAlgorithmIdentifier
```

```
ContentEncryptionAlgorithmIdentifiers ::=
  SET OF ContentEncryptionAlgorithmIdentifier
```

3.2. Unicast Messages

3.2.1. Association Messages

3.2.1.1. Message Type: "client_assoc"

This message is structured according to the NTS-Plain archetype. There is no data necessary besides that which is transported in the NTS message object, which is an ASN.1 object of type "ClientAssocData" and structured as follows:

```
ClientAssocData ::= SEQUENCE {
  nonce           NTSNonce,
  clientId        SubjectKeyIdentifier,
  digestAlgos     DigestAlgorithmIdentifiers,
  keyEncAlgos     KeyEncryptionAlgorithmIdentifiers,
  contentEncAlgos ContentEncryptionAlgorithmIdentifiers
}
```

It is identified by the following object identifier (fictional values):

```
id-clientAssocData OBJECT IDENTIFIER ::=
  {nts(TBD) association(1) clientassocdata(1)}
```


3.2.1.2. Message Type: "server_assoc"

This message is structured according to the NTS-Signed archetype. There is no data necessary besides that which is transported in the NTS message object, which is an ASN.1 object of type "ServerAssocData" and structured as follows:

```
ServerAssocData ::= SEQUENCE {
    nonce                NTSNonce,
    clientId             SubjectKeyIdentifier,
    digestAlgos          DigestAlgorithmIdentifiers,
    choiceDigestAlgo    DigestAlgorithmIdentifier,
    keyEncAlgos          KeyEncryptionAlgorithmIdentifiers,
    choiceKeyEncAlgo    KeyEncryptionAlgorithmIdentifier,
    contentEncAlgos     ContentEncryptionAlgorithmIdentifiers,
    choiceContentEncAlgo ContentEncryptionAlgorithmIdentifier
}
```

It is identified by the following object identifier (fictional values):

```
id-serverAssocData OBJECT IDENTIFIER ::=
    {nts(TBD) association(1) serverassocdata(2)}
```

3.2.2. Cookie Messages

3.2.2.1. Message Type: "client_cook"

This message is structured according to the NTS-Certified archetype. There is no data necessary besides that which is transported in the NTS message object, which is an ASN.1 object of type "ClientCookieData" and structured as follows:

```
ClientCookieData ::= SEQUENCE {
    nonce                NTSNonce,
    signAlgo             SignatureAlgorithmIdentifier,
    digestAlgo           DigestAlgorithmIdentifier,
    encAlgo              ContentEncryptionAlgorithmIdentifier,
    keyEncAlgo           KeyEncryptionAlgorithmIdentifier
}
```

It is identified by the following object identifier (fictional values):

```
id-clientCookieData OBJECT IDENTIFIER ::=
    {nts(TBD) association(1) clientcookiedata(3)}
```

3.2.2.2. Message Type: "server_cook"

This message is structured according to the "NTS-Encrypted-and-Signed" archetype. There is no data necessary besides that which is transported in the NTS message object, which is an ASN.1 object of type "ServerCookieData" and structured as follows:

```
ServerCookieData ::= SEQUENCE {
    nonce      NTSNonce,
    cookie     OCTET STRING (SIZE(16))
}
```

It is identified by the following object identifier (fictional values):

```
id-serverCookieData OBJECT IDENTIFIER ::=
    {nts(TBD) association(3) servercookiedata(4)}
```

3.2.3. Time Synchronization Messages

3.2.3.1. Message Type: "time_request"

This message is structured according to the "NTS-Plain" archetype.

This message type requires additional data to that which is included in the NTS message object, namely it requires regular time synchronization data, as an unsecured packet from a client to a server would contain. The NTS message object itself is an ASN.1 object of type "TimeRequestSecurityData", whose structure is as follows:

```
TimeRequestSecurityData ::=
SEQUENCE {
    nonce_t      NTSNonce,
    digestAlgo   DigestAlgorithmIdentifier,
    keyInputValue OCTET STRING (SIZE(16))
}
```

It is identified by the following object identifier (fictional values):

```
id-timeRequestSecurityData OBJECT IDENTIFIER ::=
    {nts(TBD) time(2) timerequestsecuritydata(1)}
```

3.2.3.2. Message Type: "time_response"

This message is also structured according to "NTS-Plain".

It requires two items of data in addition to that which is transported in the NTS message object. Like "time_request", it requires regular time synchronization data. Furthermore, it requires the Message Authentication Code (MAC) to be generated over the whole rest of the packet (including the NTS message object) and transported in some way. The NTS message object itself is an ASN.1 object of type "TimeResponseSecurityData", with the following structure:

```
TimeResponseSecurityData ::=
SEQUENCE {
    nonce_t  NTSNonce,
}
```

It is identified by the following object identifier (fictional values):

```
id-timeResponseSecurityData OBJECT IDENTIFIER ::=
    {nts(TBD) time(2) timeresponsesecuritydata(2)}
```

3.3. Broadcast Messages

3.3.1. Broadcast Parameter Messages

3.3.1.1. Message Type: "client_bpar"

This first broadcast message is structured according to the NTS-Plain archetype. There is no data necessary besides that which is transported in the NTS message object, which is an ASN.1 object of type "BroadcastParameterRequest" and structured as follows:

```
BroadcastParameterRequest ::=
SEQUENCE {
    nonce      NTSNonce,
    clientId   SubjectKeyIdentifier
}
```

It is identified by the following object identifier (fictional values):

```
id-broadcastParameterRequest OBJECT IDENTIFIER ::=
    {nts(TBD) association(1) broadcastparameterrequest(5)}
```

3.3.1.2. Message Type: "server_bpar"

This message is structured according to "NTS-Signed". There is no data necessary besides that which is transported in the NTS message object, which is an ASN.1 object of type "BroadcastParameterResponse" and structured as follows:

```
BroadcastParameterResponse ::=
SEQUENCE {
    nonce                NTSNonce,
    oneWayAlgo1          DigestAlgorithmIdentifier,
    oneWayAlgo2          DigestAlgorithmIdentifier,
    lastKey              OCTET STRING (SIZE (16)),
    intervalDuration    BIT STRING,
    disclosureDelay      INTEGER,
    nextIntervalTime    BIT STRING,
    nextIntervalIndex   INTEGER
}
```

It is identified by the following object identifier (fictional values):

```
id-broadcastParameterResponse OBJECT IDENTIFIER ::=
    {nts(TBD) association(3) broadcastparameterresponse(6)}
```

3.3.2. Broadcast Time Synchronization Message

3.3.2.1. Message Type: "server_broad"

This message is structured according to the "NTS-Plain" archetype. It requires regular broadcast time synchronization data in addition to that which is carried in the NTS message object. Like "time_response", this message type also requires a MAC, generated over all other data, to be transported within the packet. The NTS message object itself is an ASN.1 object of type "BroadcastTime". It has the following structure:

```
BroadcastTime ::=
SEQUENCE {
    thisIntervalIndex   INTEGER,
    disclosedKey        OCTET STRING (SIZE (16)),
}
```

It is identified by the following object identifier (fictional values):

```
id-broadcastTime OBJECT IDENTIFIER ::=
    {nts(TBD) time(1) broadcasttime(3)}
```

3.3.3. Broadcast Keycheck

3.3.3.1. Message Type: "client_keycheck"

This message is structured according to the "NTS-Plain" archetype. There is no data necessary besides that which is transported in the NTS message object, which is an ASN.1 object of type "ClientKeyCheckSecurityData" and structured as follows:

```
ClientKeyCheckSecurityData ::=
SEQUENCE {
    nonce_k          NTSNonce,
    interval_number  INTEGER,
    digestAlgo       DigestAlgorithmIdentifier,
    keyInputValue    OCTET STRING (SIZE(16))
}
```

It is identified by the following object identifier (fictional values):

```
id-clientKeyCheckSecurityData OBJECT IDENTIFIER ::=
    {nts(TBD) time(1) clientkeychecksecuritydata(6)}
```

3.3.3.2. Message Type: "server_keycheck"

This message is also structured according to "NTS-Plain". It requires only a MAC, generated over the NTS message object, to be included in the packet in addition to what the NTS message object itself contains. The latter is an ASN.1 object of type "ServerKeyCheckSecurityData", which is structured as follows:

```
ServerKeyCheckSecurityData ::=
SEQUENCE {
    nonce_t          NTSNonce,
    interval_number  INTEGER
}
```

It is identified by the following object identifier (fictional values):

```
id-serverKeyCheckSecurityData OBJECT IDENTIFIER ::=
    {nts(TBD) time(1) serverkeychecksecuritydata(7)}
```

4. Certificate Conventions

The syntax and processing rules for certificates are specified in [RFC5652]. In the NTS protocol, the server certificate MUST contain the following extensions:

Subject Key Identifier -- see Section 4.2.1.2 of [RFC5652].

Key Usage -- see Section 4.2.1.3 of [RFC5652].

Extended Key Usage -- see Section 4.2.1.22 of [RFC5652].

The Extended Key Usage extension MUST include the id-kp-NTSserver object identifier. When a certificate issuer includes this object identifier in the extended key usage extension, it provides an attestation that the certificate subject is a time server that supports the NTS protocol.

The id-kp-NTSserver object identifier is:

id-kp-NTSserver OBJECT IDENTIFIER ::= { TBD }

5. IANA Considerations

IANA needs to assign an object identifier for the id-kp-NTSserver key purpose and another one for the ASN.1 module in the appendix.

6. Security Considerations

To be written.

7. References

7.1. Normative References

- [ASN1] International Telecommunication Union, "Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, November 2008.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, September 2009.

7.2. Informative References

- [I-D.ietf-ntp-network-time-security] Sibold, D., Roettger, S., and K. Teichel, "Network Time Security", draft-ietf-ntp-network-time-security-08 (work in progress), March 2015.

[IEEE1588]

IEEE Instrumentation and Measurement Society. TC-9 Sensor Technology, "IEEE standard for a precision clock synchronization protocol for networked measurement and control systems", 2008.

[RFC5905]

Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

Appendix A. ASN.1 Module

The ASN.1 module contained in this appendix defines the id-kp-NTSserver object identifier.

```
NTSserverKeyPurpose
  { TBD }

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

  id-kp-NTSserver OBJECT IDENTIFIER ::= { TBD }

END
```

Authors' Addresses

Dieter Sibold
Physikalisch-Technische Bundesanstalt
Bundesallee 100
Braunschweig D-38116
Germany

Phone: +49-(0)531-592-8420
Fax: +49-531-592-698420
Email: dieter.sibold@ptb.de

Kristof Teichel
Physikalisch-Technische Bundesanstalt
Bundesallee 100
Braunschweig D-38116
Germany

Phone: +49-(0)531-592-8421
Email: kristof.teichel@ptb.de

Stephen Roettger
Google Inc.

Email: stephen.roettger@googlemail.com

Russ Housley
Vigil Security
918 Spring Knoll Drive
Herndon, VA 20170

Email: housley@vigilsec.com

NTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 7, 2016

D. Sibold
PTB
S. Roettger
Google Inc.
K. Teichel
PTB
July 06, 2015

Network Time Security
draft-ietf-ntp-network-time-security-09

Abstract

This document describes Network Time Security (NTS), a collection of measures that enable secure time synchronization with time servers using protocols like the Network Time Protocol (NTP) or the Precision Time Protocol (PTP). Its design considers the special requirements of precise timekeeping which are described in Security Requirements of Time Protocols in Packet Switched Networks [RFC7384].

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].

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 <http://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 January 7, 2016.

Copyright Notice

Copyright (c) 2015 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 (<http://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
2. Terminology	4
2.1. Terms and Abbreviations	4
2.2. Common Terminology for PTP and NTP	4
3. Security Threats	4
4. Objectives	5
5. NTS Overview	5
6. Protocol Messages	6
6.1. Unicast Time Synchronisation Messages	7
6.1.1. Preconditions for the Unicast Time Synchronization Exchange	7
6.1.2. Goals of the Unicast Time Synchronization Exchange	7
6.1.3. Message Type: "time_request"	7
6.1.4. Message Type: "time_response"	8
6.1.5. Procedure Overview of the Unicast Time Synchronization Exchange	8
6.2. Broadcast Time Synchronization Exchange	9
6.2.1. Preconditions for the Broadcast Time Synchronization Exchange	9
6.2.2. Goals of the Broadcast Time Synchronization Exchange	10
6.2.3. Message Type: "server_broad"	10
6.2.4. Procedure Overview of Broadcast Time Synchronization Exchange	11
6.3. Broadcast Keycheck	12
6.3.1. Preconditions for the Broadcast Keycheck Exchange	12
6.3.2. Goals of the Broadcast Keycheck Exchange	13
6.3.3. Message Type: "client_keycheck"	13
6.3.4. Message Type: "server_keycheck"	13
6.3.5. Procedure Overview of the Broadcast Keycheck Exchange	14
7. Server Seed Considerations	15
8. Hash Algorithms and MAC Generation	15

8.1.	Hash Algorithms	15
8.2.	MAC Calculation	16
9.	IANA Considerations	16
10.	Security Considerations	16
10.1.	Privacy	16
10.2.	Initial Verification of the Server Certificates	16
10.3.	Revocation of Server Certificates	17
10.4.	Mitigating Denial-of-Service for broadcast packets	17
10.5.	Delay Attack	17
10.6.	Random Number Generation	19
11.	Acknowledgements	19
12.	References	19
12.1.	Normative References	19
12.2.	Informative References	19
Appendix A.	(informative) TICTOC Security Requirements	20
Appendix B.	(normative) Inherent Association Protocol Messages	22
B.1.	Overview of NTS with Inherent Association Protocol	22
B.2.	Association Message Exchange	22
B.2.1.	Goals of the Association Exchange	22
B.2.2.	Message Type: "client_assoc"	23
B.2.3.	Message Type: "server_assoc"	23
B.2.4.	Procedure Overview of the Association Exchange	24
B.3.	Cookie Messages	25
B.3.1.	Goals of the Cookie Exchange	25
B.3.2.	Message Type: "client_cook"	26
B.3.3.	Message Type: "server_cook"	26
B.3.4.	Procedure Overview of the Cookie Exchange	27
B.3.5.	Broadcast Parameter Messages	28
Appendix C.	(normative) Using TESLA for Broadcast-Type Messages	30
C.1.	Server Preparation	31
C.2.	Client Preparation	32
C.3.	Sending Authenticated Broadcast Packets	33
C.4.	Authentication of Received Packets	33
Appendix D.	(informative) Dependencies	35
Authors' Addresses		37

1. Introduction

Time synchronization protocols are increasingly utilized to synchronize clocks in networked infrastructures. Successful attacks against the time synchronization protocol can seriously degrade the reliable performance of such infrastructures. Therefore, time synchronization protocols have to be secured if they are applied in environments that are prone to malicious attacks. This can be accomplished either by utilization of external security protocols, like IPsec or TLS, or by intrinsic security measures of the time synchronization protocol.

The two most popular time synchronization protocols, the Network Time Protocol (NTP) [RFC5905] and the Precision Time Protocol (PTP) [IEEE1588], currently do not provide adequate intrinsic security precautions. This document specifies security measures which enable these and possibly other protocols to verify the authenticity of the time server/master and the integrity of the time synchronization protocol packets. The utilization of these measures for a given specific time synchronization protocol has to be described in a separate document.

[RFC7384] specifies that a security mechanism for timekeeping must be designed in such a way that it does not degrade the quality of the time transfer. This implies that for time keeping the increase in bandwidth and message latency caused by the security measures should be small. Also, NTP as well as PTP work via UDP and connections are stateless on the server/master side. Therefore, all security measures in this document are designed in such a way that they add little demand for bandwidth, that the necessary calculations can be executed in a fast manner, and that the measures do not require a server/master to keep state of a connection.

2. Terminology

2.1. Terms and Abbreviations

MITM Man In The Middle

NTS Network Time Security

TESLA Timed Efficient Stream Loss-tolerant Authentication

MAC Message Authentication Code

HMAC Keyed-Hash Message Authentication Code

2.2. Common Terminology for PTP and NTP

This document refers to different time synchronization protocols, in particular to both the PTP and the NTP. Throughout the document the term "server" applies to both a PTP master and an NTP server. Accordingly, the term "client" applies to both a PTP slave and an NTP client.

3. Security Threats

The document "Security Requirements of Time Protocols in Packet Switched Networks" [RFC7384] contains a profound analysis of security threats and requirements for time synchronization protocols.

4. Objectives

The objectives of the NTS specification are as follows:

- o **Authenticity:** NTS enables the client to authenticate its time server(s).
- o **Integrity:** NTS protects the integrity of time synchronization protocol packets via a message authentication code (MAC).
- o **Confidentiality:** NTS does not provide confidentiality protection of the time synchronization packets.
- o **Authorization:** NTS enables the client to verify its time server's authorization. NTS optionally enables the server to verify the client's authorization as well.
- o **Request-Response-Consistency:** NTS enables a client to match an incoming response to a request it has sent. NTS also enables the client to deduce from the response whether its request to the server has arrived without alteration.
- o **Integration with protocols:** NTS can be used to secure different time synchronization protocols, specifically at least NTP and PTP. A client or server running an NTS-secured version of a time protocol does not negatively affect other participants who are running unsecured versions of that protocol.

5. NTS Overview

NTS initially verifies the authenticity of the time server and exchanges a symmetric key, the so-called cookie, as well as a key input value (KIV). After the cookie and the KIV are exchanged, the client then uses them to protect the authenticity and the integrity of subsequent unicast-type time synchronization packets. In order to do this, a Message Authentication Code (MAC) is attached to each time synchronization packet. The calculation of the MAC includes the whole time synchronization packet and the cookie which is shared between client and server.

The cookie is calculated according to:

$$\text{cookie} = \text{MSB}_{}(\text{HMAC}(\text{server seed}, \text{KIV})),$$

with the server seed as the key, where KIV is the client's key input value, and where the application of the function $\text{MSB}_{}$ returns only the b most significant bits. The server seed is a random value of bit length b that the server possesses, which has to remain secret.

The cookie deterministically depends on KIV as long as the server seed stays the same. The server seed has to be refreshed periodically in order to provide key freshness as required in [RFC7384]. See Section 7 for details on seed refreshing.

Since the server does not keep a state of the client, it has to recalculate the cookie each time it receives a unicast time synchronization request from the client. To this end, the client has to attach its KIV to each request (see Section 6.1).

For broadcast-type messages, authenticity and integrity of the time synchronization packets are also ensured by a MAC, which is attached to the time synchronization packet by the sender. Verification of the broadcast-type packets' authenticity is based on the TESLA protocol, in particular on its "not re-using keys" scheme, see Section 3.7.2 of [RFC4082]. TESLA uses a one-way chain of keys, where each key is the output of a one-way function applied to the previous key in the chain. The server securely shares the last element of the chain with all clients. The server splits time into intervals of uniform duration and assigns each key to an interval in reverse order. At each time interval, the server sends a broadcast packet appended by a MAC, calculated using the corresponding key, and the key of the previous disclosure interval. The client verifies the MAC by buffering the packet until disclosure of the key in its associated disclosure interval occurs. In order to be able to verify the timeliness of the packets, the client has to be loosely time synchronized with the server. This has to be accomplished before broadcast associations can be used. For checking timeliness of packets, NTS uses another, more rigorous check in addition to just the clock lookup used in the TESLA protocol. For a more detailed description of how NTS employs and customizes TESLA, see Appendix C.

6. Protocol Messages

This section describes the types of messages needed for secure time synchronization with NTS.

For some guidance on how these message types can be realized in practice, and integrated into the communication flow of existing time synchronization protocols, see [I-D.ietf-ntp-cms-for-nts-message], a companion document for NTS. Said document describes ASN.1 encodings for those message parts that have to be added to a time synchronization protocol for security reasons.

6.1. Unicast Time Synchronisation Messages

In this message exchange, the usual time synchronization process is executed, with the addition of integrity protection for all messages that the server sends. This message exchange can be repeatedly performed as often as the client desires and as long as the integrity of the server's time responses is verified successfully.

6.1.1. Preconditions for the Unicast Time Synchronization Exchange

Before this message exchange is available, there are some requirements that the client and server need to meet:

- o They MUST negotiate the hash algorithm for the MAC used in the time synchronization messages. Authenticity and integrity of the communication MUST be ensured.
- o The client MUST know a key input value KIV. Authenticity and integrity of the communication MUST be ensured.
- o Client and server MUST exchange the cookie (which depends on the KIV as described in section Section 5). Authenticity, confidentiality and integrity of the communication MUST be ensured.

One way of realising these requirements is to use the Association and Cookie Message Exchanges described in Appendix B.

6.1.2. Goals of the Unicast Time Synchronization Exchange

The unicast time synchronization exchange:

- o exchanges (unicast) time synchronization data as specified by the appropriate time synchronization protocol,
- o guarantees authenticity and integrity of the response to the client,
- o guarantees request-response-consistency to the client.

6.1.3. Message Type: "time_request"

This message is sent by the client when it requests a time exchange. It contains

- o the NTS message ID "time_request",
- o the negotiated version number,

- o a nonce,
- o the negotiated hash algorithm H,
- o the client's key input value (for which the client knows the associated cookie).

6.1.4. Message Type: "time_response"

This message is sent by the server after it has received a `time_request` message. Prior to this the server MUST recalculate the client's cookie by using the received key input value and the transmitted hash algorithm. The message contains

- o the NTS message ID "time_response",
- o the version number as transmitted in `time_request`,
- o the server's time synchronization response data,
- o the nonce transmitted in `time_request`,
- o a MAC (generated with the cookie as key) for verification of all of the above data.

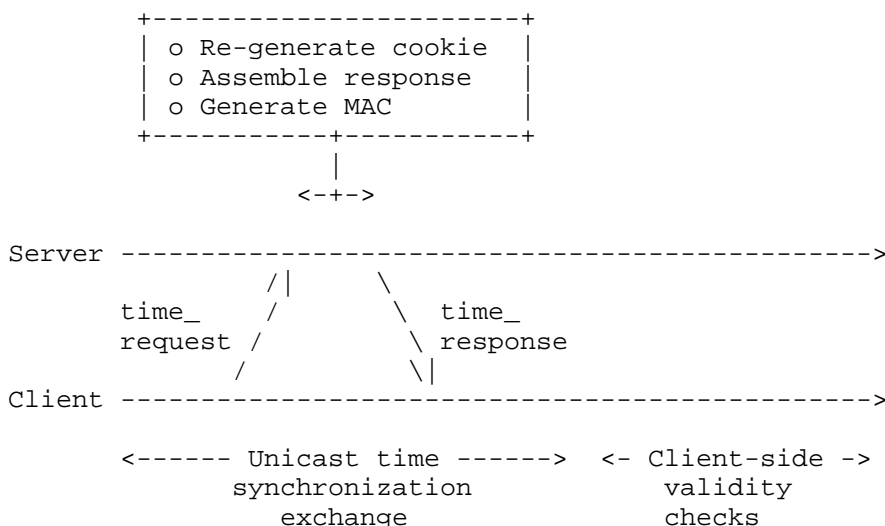
6.1.5. Procedure Overview of the Unicast Time Synchronization Exchange

For a unicast time synchronization exchange, the following steps are performed:

1. The client sends a `time_request` message to the server. The client MUST save the included nonce and the `transmit_timestamp` (from the time synchronization data) as a correlated pair for later verification steps.
2. Upon receipt of a `time_request` message, the server re-calculates the cookie, then computes the necessary time synchronization data and constructs a `time_response` message as given in Section 6.1.4.
3. The client awaits a reply in the form of a `time_response` message. Upon receipt, it checks:
 - * that the transmitted version number matches the one negotiated previously,
 - * that the transmitted nonce belongs to a previous `time_request` message,

- * that the transmit_timestamp in that time_request message matches the corresponding time stamp from the synchronization data received in the time_response, and
- * that the appended MAC verifies the received synchronization data, version number and nonce.

If at least one of the first three checks fails (i.e. if the version number does not match, if the client has never used the nonce transmitted in the time_response message, or if it has used the nonce with initial time synchronization data different from that in the response), then the client MUST ignore this time_response message. If the MAC is invalid, the client MUST do one of the following: abort the run or send another cookie request (because the cookie might have changed due to a server seed refresh). If both checks are successful, the client SHOULD continue time synchronization.



Procedure for unicast time synchronization exchange.

6.2. Broadcast Time Synchronization Exchange

6.2.1. Preconditions for the Broadcast Time Synchronization Exchange

Before this message exchange is available, there are some requirements that the client and server need to meet:

- o The client MUST receive all the information necessary to process broadcast time synchronization messages from the server. This includes
 - * the one-way functions used for building the key chain,
 - * the last key of the key chain,
 - * time interval duration,
 - * the disclosure delay (number of intervals between use and disclosure of a key),
 - * the time at which the next time interval will start, and
 - * the next interval's associated index.
- o The communication of the data listed above MUST guarantee authenticity of the server, as well as integrity and freshness of the broadcast parameters to the client.

6.2.2. Goals of the Broadcast Time Synchronization Exchange

The broadcast time synchronization exchange:

- o transmits (broadcast) time synchronization data from the server to the client as specified by the appropriate time synchronization protocol,
- o guarantees to the client that the received synchronization data has arrived in a timely manner as required by the TESLA protocol and is trustworthy enough to be stored for later checks,
- o additionally guarantees authenticity of a certain broadcast synchronization message in the client's storage.

6.2.3. Message Type: "server_broad"

This message is sent by the server over the course of its broadcast schedule. It is part of any broadcast association. It contains

- o the NTS message ID "server_broad",
- o the version number that the server is working under,
- o time broadcast data,

- o the index that belongs to the current interval (and therefore identifies the current, yet undisclosed, key),
- o the disclosed key of the previous disclosure interval (current time interval minus disclosure delay),
- o a MAC, calculated with the key for the current time interval, verifying
 - * the message ID,
 - * the version number, and
 - * the time data.

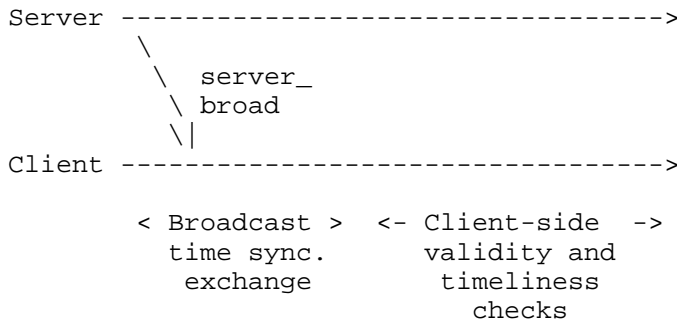
6.2.4. Procedure Overview of Broadcast Time Synchronization Exchange

A broadcast time synchronization message exchange consists of the following steps:

1. The server follows the TESLA protocol by regularly sending `server_broad` messages as described in Section 6.2.3, adhering to its own disclosure schedule.
2. The client awaits time synchronization data in the form of a `server_broadcast` message. Upon receipt, it performs the following checks:
 - * Proof that the MAC is based on a key that is not yet disclosed (packet timeliness). This is achieved via a combination of checks. First, the disclosure schedule is used, which requires loose time synchronization. If this is successful, the client obtains a stronger guarantee via a key check exchange (see below). If its timeliness is verified, the packet will be buffered for later authentication. Otherwise, the client MUST discard it. Note that the time information included in the packet will not be used for synchronization until its authenticity could also be verified.
 - * The client checks that it does not already know the disclosed key. Otherwise, the client SHOULD discard the packet to avoid a buffer overrun. If this check is successful, the client ensures that the disclosed key belongs to the one-way key chain by applying the one-way function until equality with a previous disclosed key is shown. If it is falsified, the client MUST discard the packet.

- * If the disclosed key is legitimate, then the client verifies the authenticity of any packet that it has received during the corresponding time interval. If authenticity of a packet is verified, then it is released from the buffer and its time information can be utilized. If the verification fails, then authenticity is not given. In this case, the client MUST request authentic time from the server by means other than broadcast messages. Also, the client MUST re-initialize the broadcast sequence with a "client_bpar" message if the one-way key chain expires, which it can check via the disclosure schedule.

See RFC 4082[RFC4082] for a detailed description of the packet verification process.



Procedure for broadcast time synchronization exchange.

6.3. Broadcast Keycheck

This message exchange is performed for an additional check of packet timeliness in the course of the TESLA scheme, see Appendix C.

6.3.1. Preconditions for the Broadcast Keycheck Exchange

Before this message exchange is available, there are some requirements that the client and server need to meet:

- o They MUST negotiate the hash algorithm for the MAC used in the time synchronization messages. Authenticity and integrity of the communication MUST be ensured.
- o The client MUST know a key input value KIV. Authenticity and integrity of the communication MUST be ensured.
- o Client and server MUST exchange the cookie (which depends on the KIV as described in section Section 5). Authenticity,

confidentiality and integrity of the communication MUST be ensured.

These requirements conform to those for the unicast time synchronization exchange. Accordingly, they too can be realised via the Association and Cookie Message Exchanges described in Appendix B (Appendix B).

6.3.2. Goals of the Broadcast Keycheck Exchange

The keycheck exchange:

- o guarantees to the client that the key belonging to the respective TESLA interval communicated in the exchange had not been disclosed before the `client_keycheck` message was sent.
- o guarantees to the client the timeliness of any broadcast packet secured with this key if it arrived before `client_keycheck` was sent.

6.3.3. Message Type: "client_keycheck"

A message of this type is sent by the client in order to initiate an additional check of packet timeliness for the TESLA scheme. It contains

- o the NTS message ID "client_keycheck",
- o the NTS version number negotiated during association,
- o a nonce,
- o an interval number from the TESLA disclosure schedule,
- o the hash algorithm H negotiated during association, and
- o the client's key input value KIV.

6.3.4. Message Type: "server_keycheck"

A message of this type is sent by the server upon receipt of a `client_keycheck` message during the broadcast loop of the server. Prior to this, the server MUST recalculate the client's cookie by using the received key input value and the transmitted hash algorithm. It contains

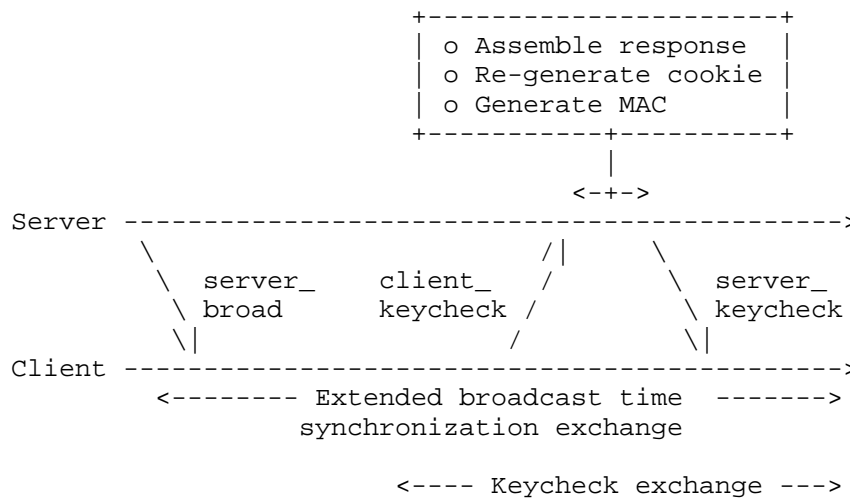
- o the NTS message ID "server_keycheck"

- o the version number as transmitted in "client_keycheck,
- o the nonce transmitted in the client_keycheck message,
- o the interval number transmitted in the client_keycheck message,
and
- o a MAC (generated with the cookie as key) for verification of all
of the above data.

6.3.5. Procedure Overview of the Broadcast Keycheck Exchange

A broadcast keycheck message exchange consists of the following steps:

1. The client sends a client_keycheck message. It MUST memorize the nonce and the time interval number that it sends as a correlated pair.
2. Upon receipt of a client_keycheck message, the server looks up whether it has already disclosed the key associated with the interval number transmitted in that message. If it has not disclosed it, it constructs and sends the appropriate server_keycheck message as described in Section 6.3.4. For more details, see also Appendix C.
3. The client awaits a reply in the form of a server_keycheck message. On receipt, it performs the following checks:
 - * that the transmitted version number matches the one negotiated previously,
 - * that the transmitted nonce belongs to a previous client_keycheck message,
 - * that the TESLA interval number in that client_keycheck message matches the corresponding interval number from the server_keycheck, and
 - * that the appended MAC verifies the received data.



Procedure for extended broadcast time synchronization exchange.

7. Server Seed Considerations

The server has to calculate a random seed which has to be kept secret. The server MUST generate a seed for each supported hash algorithm, see Section 8.1.

According to the requirements in [RFC7384], the server MUST refresh each server seed periodically. Consequently, the cookie memorized by the client becomes obsolete. In this case, the client cannot verify the MAC attached to subsequent time response messages and has to respond accordingly by re-initiating the protocol with a cookie request (Appendix B.3).

8. Hash Algorithms and MAC Generation

8.1. Hash Algorithms

Hash algorithms are used for calculation of the cookie and the MAC. The client and the server negotiate a hash algorithm H during the association phase at the beginning. The selected algorithm H is used for all hashing processes in that run.

In the TESLA scheme, hash algorithms are used as pseudo-random functions to construct the one-way key chain. Here, the utilized hash algorithm is communicated by the server and is non-negotiable.

Note:

Any hash algorithm is prone to be compromised in the future. A successful attack on a hash algorithm would enable any NTS client to derive the server seed from its own cookie. Therefore, the server MUST have separate seed values for its different supported hash algorithms. This way, knowledge gained from an attack on a hash algorithm H can at least only be used to compromise such clients who use hash algorithm H as well.

8.2. MAC Calculation

For the calculation of the MAC, client and server use a Keyed-Hash Message Authentication Code (HMAC) approach [RFC2104]. The HMAC is generated with the hash algorithm specified by the client (see Section 8.1).

9. IANA Considerations

10. Security Considerations

10.1. Privacy

The payload of time synchronization protocol packets of two-way time transfer approaches like NTP and PTP consists basically of time stamps, which are not considered secret [RFC7384]. Therefore, encryption of the time synchronization protocol packet's payload is not considered in this document. However, an attacker can exploit the exchange of time synchronization protocol packets for topology detection and inference attacks as described in [I-D.iab-privsec-confidentiality-threat]. To make such attacks more difficult, that draft recommends the encryption of the packet payload. Yet, in the case of time synchronization protocols the confidentiality protection of time synchronization packet's payload is of secondary importance since the packet's meta data (IP addresses, port numbers, possibly packet size and regular sending intervals) carry more information than the payload. To enhance the privacy of the time synchronization partners, the usage of tunnel protocols such as IPsec and MACsec, where applicable, is therefore more suited than confidentiality protection of the payload.

10.2. Initial Verification of the Server Certificates

The client may wish to verify the validity of certificates during the initial association phase. Since it generally has no reliable time during this initial communication phase, it is impossible to verify the period of validity of the certificates. To solve this chicken-and-egg problem, the client has to rely on external means.

10.3. Revocation of Server Certificates

According to Section 7, it is the client's responsibility to initiate a new association with the server after the server's certificate expires. To this end, the client reads the expiration date of the certificate during the certificate message exchange (Appendix B.2.3). Furthermore, certificates may also be revoked prior to the normal expiration date. To increase security the client MAY periodically verify the state of the server's certificate via Online Certificate Status Protocol (OCSP) Online Certificate Status Protocol (OCSP) [RFC6960].

10.4. Mitigating Denial-of-Service for broadcast packets

TESLA authentication buffers packets for delayed authentication. This makes the protocol vulnerable to flooding attacks, causing the client to buffer excessive numbers of packets. To add stronger DoS protection to the protocol, the client and the server use the "not re-using keys" scheme of TESLA as pointed out in Section 3.7.2 of RFC 4082 [RFC4082]. In this scheme the server never uses a key for the MAC generation more than once. Therefore, the client can discard any packet that contains a disclosed key it already knows, thus preventing memory flooding attacks.

Discussion: Note that an alternative approach to enhance TESLA's resistance against DoS attacks involves the addition of a group MAC to each packet. This requires the exchange of an additional shared key common to the whole group. This adds additional complexity to the protocol and hence is currently not considered in this document.

10.5. Delay Attack

In a packet delay attack, an adversary with the ability to act as a MITM delays time synchronization packets between client and server asymmetrically [RFC7384]. This prevents the client from accurately measuring the network delay, and hence its time offset to the server [Mizrahi]. The delay attack does not modify the content of the exchanged synchronization packets. Therefore, cryptographic means do not provide a feasible way to mitigate this attack. However, several non-cryptographic precautions can be taken in order to detect this attack.

1. Usage of multiple time servers: this enables the client to detect the attack, provided that the adversary is unable to delay the synchronization packets between the majority of servers. This approach is commonly used in NTP to exclude incorrect time servers [RFC5905].

2. Multiple communication paths: The client and server utilize different paths for packet exchange as described in the I-D [I-D.ietf-tictoc-multi-path-synchronization]. The client can detect the attack, provided that the adversary is unable to manipulate the majority of the available paths [Shpiner]. Note that this approach is not yet available, neither for NTP nor for PTP.
3. Usage of an encrypted connection: the client exchanges all packets with the time server over an encrypted connection (e.g. IPsec). This measure does not mitigate the delay attack, but it makes it more difficult for the adversary to identify the time synchronization packets.
4. For unicast-type messages: Introduction of a threshold value for the delay time of the synchronization packets. The client can discard a time server if the packet delay time of this time server is larger than the threshold value.

Additional provision against delay attacks has to be taken for broadcast-type messages. This mode relies on the TESLA scheme which is based on the requirement that a client and the broadcast server are loosely time synchronized. Therefore, a broadcast client has to establish time synchronization with its broadcast server before it starts utilizing broadcast messages for time synchronization.

One possible way to achieve this initial synchronization is to establish a unicast association with its broadcast server until time synchronization and calibration of the packet delay time is achieved. After that, the client can establish a broadcast association with the broadcast server and utilizes TESLA to verify integrity and authenticity of any received broadcast packets.

An adversary who is able to delay broadcast packets can cause a time adjustment at the receiving broadcast clients. If the adversary delays broadcast packets continuously, then the time adjustment will accumulate until the loose time synchronization requirement is violated, which breaks the TESLA scheme. To mitigate this vulnerability the security condition in TESLA has to be supplemented by an additional check in which the client, upon receipt of a broadcast message, verifies the status of the corresponding key via a unicast message exchange with the broadcast server (see Appendix C.4 for a detailed description of this check). Note that a broadcast client should also apply the above-mentioned precautions as far as possible.

10.6. Random Number Generation

At various points of the protocol, the generation of random numbers is required. The employed methods of generation need to be cryptographically secure. See [RFC4086] for guidelines concerning this topic.

11. Acknowledgements

The authors would like to thank Tal Mizrahi, Russ Housley, Steven Bellovin, David Mills, Kurt Roeckx, Rainer Bermbach, Martin Langer and Florian Weimer for discussions and comments on the design of NTS. Also, thanks go to Harlan Stenn for his technical review and specific text contributions to this document.

12. References

12.1. Normative References

- [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4082] Perrig, A., Song, D., Canetti, R., Tygar, J., and B. Briscoe, "Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction", RFC 4082, June 2005.
- [RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, October 2014.

12.2. Informative References

- [I-D.iab-privsec-confidentiality-threat]
Barnes, R., Schneier, B., Jennings, C., Hardie, T., Trammell, B., Huitema, C., and D. Borkmann,
"Confidentiality in the Face of Pervasive Surveillance: A Threat Model and Problem Statement", draft-iab-privsec-confidentiality-threat-07 (work in progress), May 2015.
- [I-D.ietf-ntp-cms-for-nts-message]
Sibold, D., Teichel, K., Roettger, S., and R. Housley,
"Protecting Network Time Security Messages with the Cryptographic Message Syntax (CMS)", draft-ietf-ntp-cms-for-nts-message-03 (work in progress), April 2015.

- [I-D.ietf-tictoc-multi-path-synchronization]
Shpiner, A., Tse, R., Schelp, C., and T. Mizrahi, "Multi-Path Time Synchronization", draft-ietf-tictoc-multi-path-synchronization-02 (work in progress), April 2015.
- [IEEE1588]
IEEE Instrumentation and Measurement Society. TC-9 Sensor Technology, "IEEE standard for a precision clock synchronization protocol for networked measurement and control systems", 2008.
- [Mizrahi] Mizrahi, T., "A game theoretic analysis of delay attacks against time synchronization protocols", in Proceedings of Precision Clock Synchronization for Measurement Control and Communication, ISPCS 2012, pp. 1-6, September 2012.
- [RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.
- [RFC5905] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.
- [RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, June 2013.
- [Shpiner] Shpiner, A., Revah, Y., and T. Mizrahi, "Multi-path Time Protocols", in Proceedings of Precision Clock Synchronization for Measurement Control and Communication, ISPCS 2013, pp. 1-6, September 2013.

Appendix A. (informative) TICTOC Security Requirements

The following table compares the NTS specifications against the TICTOC security requirements [RFC7384].

Section	Requirement from RFC 7384	Requirement level	NTS
5.1.1	Authentication of Servers	MUST	OK
5.1.1	Authorization of Servers	MUST	OK
5.1.2	Recursive Authentication of Servers (Stratum 1)	MUST	OK

5.1.2	Recursive Authorization of Servers (Stratum 1)	MUST	OK
5.1.3	Authentication and Authorization of Clients	MAY	Optional, Limited
5.2	Integrity protection	MUST	OK
5.3	Spoofing Prevention	MUST	OK
5.4	Protection from DoS attacks against the time protocol	SHOULD	OK
5.5	Replay protection	MUST	OK
5.6	Key freshness	MUST	OK
	Security association	SHOULD	OK
	Unicast and multicast associations	SHOULD	OK
5.7	Performance: no degradation in quality of time transfer	MUST	OK
	Performance: lightweight computation	SHOULD	OK
	Performance: storage	SHOULD	OK
	Performance: bandwidth	SHOULD	OK
5.8	Confidentiality protection	MAY	NO
5.9	Protection against Packet Delay and Interception Attacks	MUST	Limited*)
5.10	Secure mode	MUST	OK
	Hybrid mode	SHOULD	-

*) See discussion in Section 10.5.

Comparison of NTS specification against Security Requirements of Time Protocols in Packet Switched Networks (RFC 7384)

Appendix B. (normative) Inherent Association Protocol Messages

One option for completing association, cookie exchange, and also broadcast parameter exchange between a client and server is to use the message exchanges listed below.

B.1. Overview of NTS with Inherent Association Protocol

This inherent association protocol applies X.509 certificates to verify the authenticity of the time server and to exchange the cookie. This is done in two separate message exchanges, described below. A client needs a public/private key pair for encryption, with the public key enclosed in a certificate. A server needs a public/private key pair for signing, with the public key enclosed in a certificate. If a participant intends to act as both a client and a server, it **MUST** have two different key pairs for these purposes.

If this protocol is employed, the hash value of the client's certificate is used as the client's key input value, i.e. the cookie is calculated according to:

$$\text{cookie} = \text{MSB_} \langle b \rangle (\text{HMAC}(\text{server seed}, \text{H}(\text{certificate of client}))).$$

The client's certificate contains the client's public key and enables the server to identify the client, if client authorization is desired.

B.2. Association Message Exchange

In this message exchange, the participants negotiate the hash and encryption algorithms that are used throughout the protocol. In addition, the client receives the certification chain up to a trusted anchor. With the established certification chain the client is able to verify the server's signatures and, hence, the authenticity of future NTS messages from the server is ensured.

B.2.1. Goals of the Association Exchange

The association exchange:

- o enables the client to verify any communication with the server as authentic,
- o lets the participants negotiate NTS version and algorithms,
- o guarantees authenticity and integrity of the negotiation result to the client,

- o guarantees to the client that the negotiation result is based on the client's original, unaltered request.

B.2.2. Message Type: "client_assoc"

The protocol sequence starts with the client sending an association message, called `client_assoc`. This message contains

- o the NTS message ID "client_assoc",
- o a nonce,
- o the version number of NTS that the client wants to use (this SHOULD be the highest version number that it supports),
- o the hostname of the client,
- o a selection of accepted hash algorithms, and
- o a selection of accepted encryption algorithms.

B.2.3. Message Type: "server_assoc"

This message is sent by the server upon receipt of `client_assoc`. It contains

- o the NTS message ID "server_assoc",
- o the nonce transmitted in `client_assoc`,
- o the client's proposal for the version number, selection of accepted hash algorithms and selection of accepted encryption algorithms, as transmitted in `client_assoc`,
- o the version number used for the rest of the protocol (which SHOULD be determined as the minimum over the client's suggestion in the `client_assoc` message and the highest supported by the server),
- o the hostname of the server,
- o the server's choice of algorithm for encryption and for cryptographic hashing, all of which MUST be chosen from the client's proposals,
- o a signature, calculated over the data listed above, with the server's private key and according to the signature algorithm which is also used for the certificates that are included (see below), and

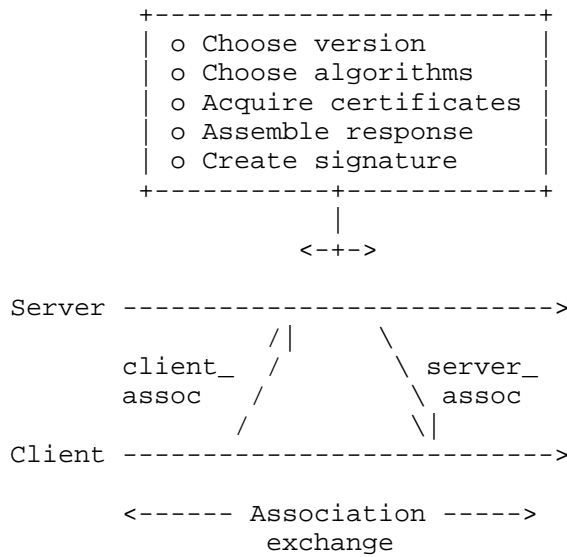
- o a chain of certificates, which starts at the server and goes up to a trusted authority; each certificate MUST be certified by the one directly following it.

B.2.4. Procedure Overview of the Association Exchange

For an association exchange, the following steps are performed:

1. The client sends a `client_assoc` message to the server. It MUST keep the transmitted values for the version number and algorithms available for later checks.
2. Upon receipt of a `client_assoc` message, the server constructs and sends a reply in the form of a `server_assoc` message as described in Appendix B.2.3. Upon unsuccessful negotiation for version number or algorithms the `server_assoc` message MUST contain an error code.
3. The client waits for a reply in the form of a `server_assoc` message. After receipt of the message it performs the following checks:
 - * The client checks that the message contains a conforming version number.
 - * It checks that the nonce sent back by the server matches the one transmitted in `client_assoc`,
 - * It also verifies that the server has chosen the encryption and hash algorithms from its proposal sent in the `client_assoc` message and that this proposal was not altered.
 - * Furthermore, it performs authenticity checks on the certificate chain and the signature.

If one of the checks fails, the client MUST abort the run.



Procedure for association and cookie exchange.

B.3. Cookie Messages

During this message exchange, the server transmits a secret cookie to the client securely. The cookie will later be used for integrity protection during unicast time synchronization.

B.3.1. Goals of the Cookie Exchange

The cookie exchange:

- o enables the server to check the client's authorization via its certificate (optional),
- o supplies the client with the correct cookie and corresponding KIV for its association to the server,
- o guarantees to the client that the cookie originates from the server and that it is based on the client's original, unaltered request.
- o guarantees that the received cookie is unknown to anyone but the server and the client.

B.3.2. Message Type: "client_cook"

This message is sent by the client upon successful authentication of the server. In this message, the client requests a cookie from the server. The message contains

- o the NTS message ID "client_cook",
- o a nonce,
- o the negotiated version number,
- o the negotiated signature algorithm,
- o the negotiated encryption algorithm,
- o the negotiated hash algorithm H,
- o the client's certificate.

B.3.3. Message Type: "server_cook"

This message is sent by the server upon receipt of a client_cook message. The server generates the hash of the client's certificate, as conveyed during client_cook, in order to calculate the cookie according to Section 5. This message contains

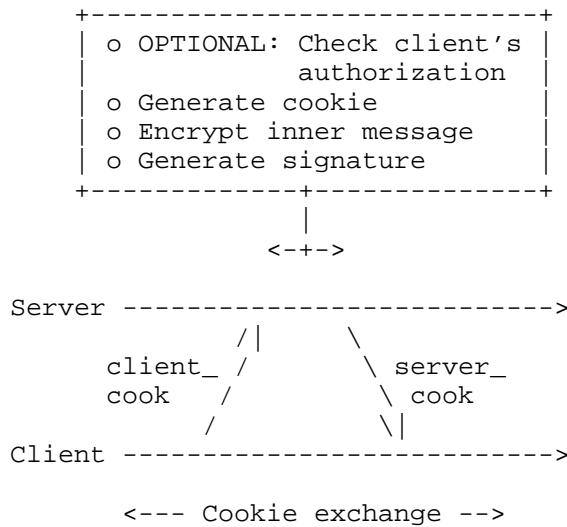
- o the NTS message ID "server_cook"
- o the version number as transmitted in client_cook,
- o a concatenated datum which is encrypted with the client's public key, according to the encryption algorithm transmitted in the client_cook message. The concatenated datum contains
 - * the nonce transmitted in client_cook, and
 - * the cookie.
- o a signature, created with the server's private key, calculated over all of the data listed above. This signature MUST be calculated according to the transmitted signature algorithm from the client_cook message.

B.3.4. Procedure Overview of the Cookie Exchange

For a cookie exchange, the following steps are performed:

1. The client sends a `client_cook` message to the server. The client MUST save the included nonce until the reply has been processed.
2. Upon receipt of a `client_cook` message, the server checks whether it supports the given cryptographic algorithms. It then calculates the cookie according to the formula given in Section 5. The server MAY use the client's certificate to check that the client is authorized to use the secure time synchronization service. With this, it MUST construct a `server_cook` message as described in Appendix B.3.3.
3. The client awaits a reply in the form of a `server_cook` message; upon receipt it executes the following actions:
 - * It verifies that the received version number matches the one negotiated beforehand.
 - * It verifies the signature using the server's public key. The signature has to authenticate the encrypted data.
 - * It decrypts the encrypted data with its own private key.
 - * It checks that the decrypted message is of the expected format: the concatenation of a nonce and a cookie of the expected bit lengths.
 - * It verifies that the received nonce matches the nonce sent in the `client_cook` message.

If one of those checks fails, the client MUST abort the run.



Procedure for association and cookie exchange.

B.3.5. Broadcast Parameter Messages

In this message exchange, the client receives the necessary information to execute the TESLA protocol in a secured broadcast association. The client can only initiate a secure broadcast association after successful association and cookie exchanges and only if it has made sure that its clock is roughly synchronized to the server's.

See Appendix C for more details on TESLA.

B.3.5.1. Goals of the Broadcast Parameter Exchange

The broadcast parameter exchange

- o provides the client with all the information necessary to process broadcast time synchronization messages from the server, and
- o guarantees authenticity, integrity and freshness of the broadcast parameters to the client.

B.3.5.2. Message Type: "client_bpar"

This message is sent by the client in order to establish a secured time broadcast association with the server. It contains

- o the NTS message ID "client_bpar",

- o the NTS version number negotiated during association,
- o a nonce,
- o the client's hostname, and
- o the signature algorithm negotiated during association.

B.3.5.3. Message Type: "server_bpar"

This message is sent by the server upon receipt of a client_bpar message during the broadcast loop of the server. It contains

- o the NTS message ID "server_bpar",
- o the version number as transmitted in the client_bpar message,
- o the nonce transmitted in client_bpar,
- o the one-way functions used for building the key chain, and
- o the disclosure schedule of the keys. This contains:
 - * the last key of the key chain,
 - * time interval duration,
 - * the disclosure delay (number of intervals between use and disclosure of a key),
 - * the time at which the next time interval will start, and
 - * the next interval's associated index.
- o The message also contains a signature signed by the server with its private key, verifying all the data listed above.

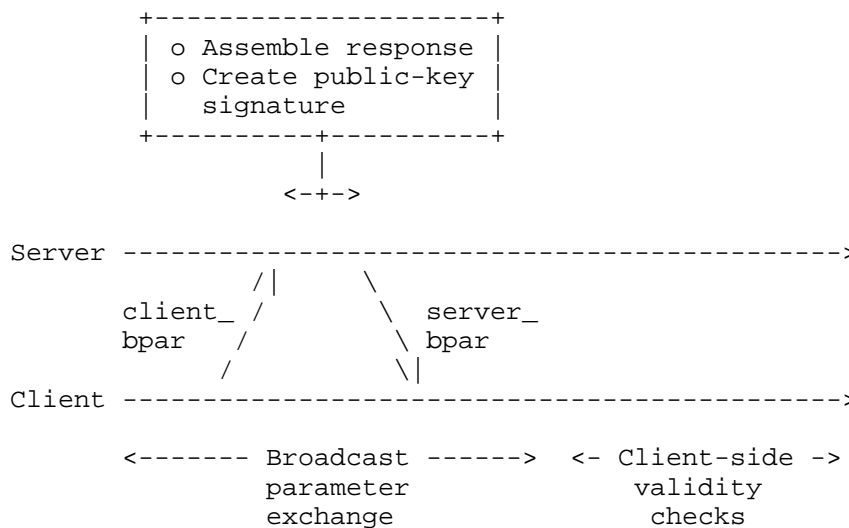
B.3.5.4. Procedure Overview of the Broadcast Parameter Exchange

A broadcast parameter exchange consists of the following steps:

1. The client sends a client_bpar message to the server. It MUST remember the transmitted values for the nonce, the version number and the signature algorithm.
2. Upon receipt of a client_bpar message, the server constructs and sends a server_bpar message as described in Appendix B.3.5.3.

3. The client waits for a reply in the form of a `server_bpar` message, on which it performs the following checks:
 - * The message must contain all the necessary information for the TESLA protocol, as listed in Appendix B.3.5.3.
 - * The message must contain a nonce belonging to a `client_bpar` message that the client has previously sent.
 - * Verification of the message's signature.

If any information is missing or if the server's signature cannot be verified, the client MUST abort the broadcast run. If all checks are successful, the client MUST remember all the broadcast parameters received for later checks.



Procedure for unicast time synchronization exchange.

Appendix C. (normative) Using TESLA for Broadcast-Type Messages

For broadcast-type messages, NTS adopts the TESLA protocol with some customizations. This appendix provides details on the generation and usage of the one-way key chain collected and assembled from [RFC4082]. Note that NTS uses the "not re-using keys" scheme of TESLA as described in Section 3.7.2. of [RFC4082].

C.1. Server Preparation

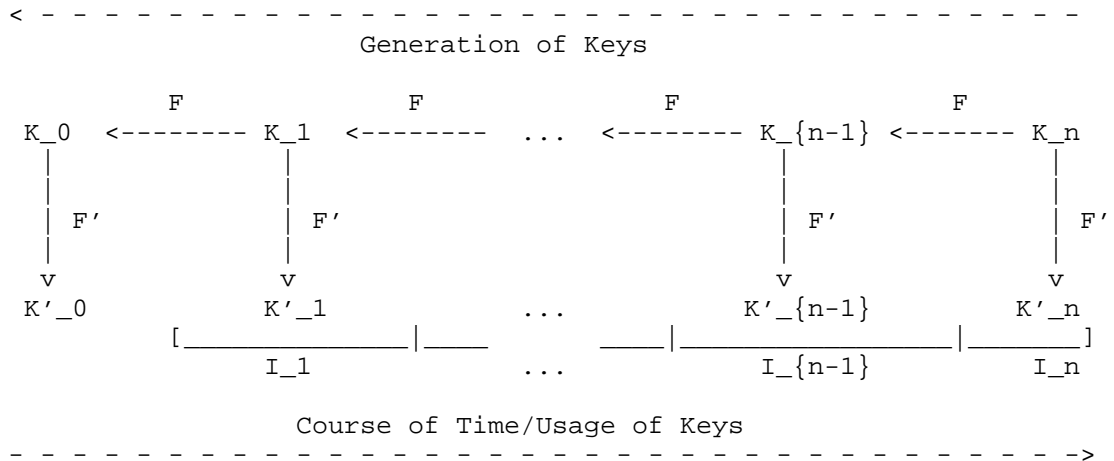
Server setup:

1. The server determines a reasonable upper bound B on the network delay between itself and an arbitrary client, measured in milliseconds.
2. It determines the number $n+1$ of keys in the one-way key chain. This yields the number n of keys that are usable to authenticate broadcast packets. This number n is therefore also the number of time intervals during which the server can send authenticated broadcast messages before it has to calculate a new key chain.
3. It divides time into n uniform intervals I_1, I_2, \dots, I_n . Each of these time intervals has length L , measured in milliseconds. In order to fulfill the requirement 3.7.2. of RFC 4082, the time interval L has to be shorter than the time interval between the broadcast messages.
4. The server generates a random key K_n .
5. Using a one-way function F , the server generates a one-way chain of $n+1$ keys $K_0, K_1, \dots, K_{\{n\}}$ according to
$$K_i = F(K_{\{i+1\}}).$$
6. Using another one-way function F' , it generates a sequence of n MAC keys $K'_0, K'_1, \dots, K'_{\{n-1\}}$ according to
$$K'_i = F'(K_i).$$
7. Each MAC key K'_i is assigned to the time interval I_i .
8. The server determines the key disclosure delay d , which is the number of intervals between using a key and disclosing it. Note that although security is provided for all choices $d > 0$, the choice still makes a difference:
 - * If d is chosen too short, the client might discard packets because it fails to verify that the key used for its MAC has not yet been disclosed.
 - * If d is chosen too long, the received packets have to be buffered for an unnecessarily long time before they can be verified by the client and be subsequently utilized for time synchronization.

It is RECOMMENDED that the server calculate d according to

$$d = \text{ceil}(2*B / L) + 1,$$

where `ceil` yields the smallest integer greater than or equal to its argument.



A schematic explanation of the TESLA protocol's one-way key chain

C.2. Client Preparation

A client needs the following information in order to participate in a TESLA broadcast:

- o One key K_i from the one-way key chain, which has to be authenticated as belonging to the server. Typically, this will be K_0 .
- o The disclosure schedule of the keys. This consists of:
 - * the length n of the one-way key chain,
 - * the length L of the time intervals I_1, I_2, \dots, I_n ,
 - * the starting time T_i of an interval I_i . Typically this is the starting time T_1 of the first interval;
 - * the disclosure delay d .
- o The one-way function F used to recursively derive the keys in the one-way key chain,

- o The second one-way function F' used to derive the MAC keys K'_0, K'_1, \dots, K'_n from the keys in the one-way chain.
- o An upper bound D_t on how far its own clock is "behind" that of the server.

Note that if D_t is greater than $(d - 1) * L$, then some authentic packets might be discarded. If D_t is greater than $d * L$, then all authentic packets will be discarded. In the latter case, the client SHOULD NOT participate in the broadcast, since there will be no benefit in doing so.

C.3. Sending Authenticated Broadcast Packets

During each time interval I_i , the server sends at most one authenticated broadcast packet P_i . Such a packet consists of:

- o a message M_i ,
- o the index i (in case a packet arrives late),
- o a MAC authenticating the message M_i , with K'_i used as key,
- o the key $K_{\{i-d\}}$, which is included for disclosure.

C.4. Authentication of Received Packets

When a client receives a packet P_i as described above, it first checks that it has not already received a packet with the same disclosed key. This is done to avoid replay/flooding attacks. A packet that fails this test is discarded.

Next, the client begins to check the packet's timeliness by ensuring that according to the disclosure schedule and with respect to the upper bound D_t determined above, the server cannot have disclosed the key K_i yet. Specifically, it needs to check that the server's clock cannot read a time that is in time interval $I_{\{i+d\}}$ or later. Since it works under the assumption that the server's clock is not more than D_t "ahead" of the client's clock, the client can calculate an upper bound t_i for the server's clock at the time when P_i arrived. This upper bound t_i is calculated according to

$$t_i = R + D_t,$$

where R is the client's clock at the arrival of P_i . This implies that at the time of arrival of P_i , the server could have been in interval I_x at most, with

$$x = \text{floor}((t_i - T_1) / L) + 1,$$

where floor gives the greatest integer less than or equal to its argument. The client now needs to verify that

$$x < i+d$$

is valid (see also Section 3.5 of [RFC4082]). If it is falsified, it is discarded.

If the check above is successful, the client performs another more rigorous check: it sends a key check request to the server (in the form of a client_keycheck message), asking explicitly if K_i has already been disclosed. It remembers the time stamp t_{check} of the sending time of that request as well as the nonce it used correlated with the interval number i . If it receives an answer from the server stating that K_i has not yet been disclosed and it is able to verify the HMAC on that response, then it deduces that K_i was undisclosed at t_{check} and therefore also at R . In this case, the client accepts P_i as timely.

Next the client verifies that a newly disclosed key $K_{\{i-d\}}$ belongs to the one-way key chain. To this end, it applies the one-way function F to $K_{\{i-d\}}$ until it can verify the identity with an earlier disclosed key (see Clause 3.5 in RFC 4082, item 3).

Next the client verifies that the transmitted time value s_i belongs to the time interval I_i , by checking

$$T_i \leq s_i, \text{ and}$$

$$s_i < T_{\{i+1\}}.$$

If it is falsified, the packet MUST be discarded and the client MUST reinitialize its broadcast module by performing time synchronization by other means than broadcast messages, and it MUST perform a new broadcast parameter exchange (because a falsification of this check yields that the packet was not generated according to protocol, which suggests an attack).

If a packet P_i passes all the tests listed above, it is stored for later authentication. Also, if at this time there is a package with index $i-d$ already buffered, then the client uses the disclosed key $K_{\{i-d\}}$ to derive $K'_{\{i-d\}}$ and uses that to check the MAC included in package $P_{\{i-d\}}$. Upon success, it regards $M_{\{i-d\}}$ as authenticated.

Appendix D. (informative) Dependencies

Issuer	Type	Owner	Description
Server PKI	private key (signature)	server	Used for server_assoc, server_cook, server_bpar.
	public key (signature)	client	The server uses the private key to sign these messages. The client uses the public key to verify them.
	certificate	server	The certificate is used in server_assoc messages, for verifying authentication and (optionally) authorization.
Client PKI	private key (encryption)	client	The server uses the client's public key to encrypt the content of server_cook
	public key (encryption)	server	messages. The client uses the private key to decrypt them. The certificate is
	certificate	client	sent in client_cook messages, where it is used for trans- portation of the public key as well as (optionally) for verification of client authorization.

Authors' Addresses

Dieter Sibold
Physikalisch-Technische Bundesanstalt
Bundesallee 100
Braunschweig D-38116
Germany

Phone: +49-(0)531-592-8420
Fax: +49-531-592-698420
Email: dieter.sibold@ptb.de

Stephen Roettger
Google Inc.

Email: stephen.roettger@googlemail.com

Kristof Teichel
Physikalisch-Technische Bundesanstalt
Bundesallee 100
Braunschweig D-38116
Germany

Phone: +49-(0)531-592-8421
Email: kristof.teichel@ptb.de

NTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 7, 2016

D. Sibold
PTB
S. Roettger
Google Inc
K. Teichel
PTB
July 06, 2015

Using the Network Time Security Specification to Secure the Network Time
Protocol
draft-ietf-ntp-using-nts-for-ntp-01

Abstract

This document describes how to use the measures described in the Network Time Security (NTS) specification to secure time synchronization with servers using the Network Time Protocol (NTP).

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].

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 <http://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 January 7, 2016.

Copyright Notice

Copyright (c) 2015 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 (<http://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
2. Objectives	3
3. Terms and Abbreviations	4
4. Overview of NTS-Secured NTP	4
4.1. Symmetric and Client/Server Mode	4
4.2. Broadcast Mode	4
5. Protocol Sequence	5
5.1. The Client	5
5.1.1. The Client in Unicast Mode	5
5.1.2. The Client in Broadcast Mode	7
5.2. The Server	9
5.2.1. The Server in Unicast Mode	9
5.2.2. The Server in Broadcast Mode	9
6. Implementation Notes: ASN.1 Structures and Use of the CMS	10
6.1. Unicast Messages	10
6.1.1. Association Messages	10
6.1.2. Cookie Messages	11
6.1.3. Time Synchronization Messages	11
6.2. Broadcast Messages	12
6.2.1. Broadcast Parameter Messages	12
6.2.2. Broadcast Time Synchronization Message	12
6.2.3. Broadcast Keycheck	12
7. Security Considerations	13
7.1. Employing Alternative Means for Association and Cookie Exchange	13
7.2. Usage of NTP Pools	13
7.3. Server Seed Lifetime	13
7.4. Supported Hash Algorithms	13
8. Acknowledgements	13
9. References	13
9.1. Normative References	13
9.2. Informative References	14
Appendix A. Flow Diagrams of Client Behaviour	14
Appendix B. Bit Lengths for Employed Primitives	17
Authors' Addresses	17

1. Introduction

One of the most popular time synchronization protocols, the Network Time Protocol (NTP) [RFC5905], currently does not provide adequate intrinsic security precautions. The Network Time Security draft [I-D.ietf-ntp-network-time-security] specifies security measures which can be used to enable time synchronization protocols to verify authenticity of the time server and integrity of the time synchronization protocol packets.

This document provides detail on how to specifically use those measures to secure time synchronization between NTP clients and servers.

2. Objectives

The objectives of the NTS specification are as follows:

- o **Authenticity:** NTS enables an NTP client to authenticate its time server(s).
- o **Integrity:** NTS protects the integrity of NTP time synchronization protocol packets via a message authentication code (MAC).
- o **Confidentiality:** NTS does not provide confidentiality protection of the time synchronization packets.
- o **Authorization:** NTS optionally enables the server to verify the client's authorization.
- o **Request-Response-Consistency:** NTS enables a client to match an incoming response to a request it has sent. NTS also enables the client to deduce from the response whether its request to the server has arrived without alteration.
- o **Modes of operation:** Both the unicast and the broadcast mode of NTP are supported.
- o **Hybrid mode:** Both secure and insecure communication modes are possible for both NTP servers and clients.
- o **Compatibility:**
 - * Unsecured NTP associations are not affected.
 - * An NTP server that does not support NTS is not affected by NTS-secured authentication requests.

3. Terms and Abbreviations

CMS Cryptographic Message Syntax [RFC5652]
HMAC Keyed-Hash Message Authentication Code
MAC Message Authentication Code
MITM Man In The Middle
NTP Network Time Protocol [RFC5905]
NTS Network Time Security
TESLA Timed Efficient Stream Loss-Tolerant Authentication [RFC4082]

4. Overview of NTS-Secured NTP

4.1. Symmetric and Client/Server Mode

The server does not keep a state of the client. NTS initially verifies the authenticity of the time server and exchanges a symmetric key, the so-called cookie and a key input value (KIV). The "association" and "cookie" message exchanges described in [I-D.ietf-ntp-network-time-security], Appendix B., can be utilized for the exchange of the cookie and KIV. An implementation MUST support the use of these exchanges. It MAY additionally support the use of any alternative secure communication for this purpose, as long as it fulfills the preconditions given in [I-D.ietf-ntp-network-time-security], Section 6.1.1.

After the cookie and KIV are exchanged, the participants then use them to protect the authenticity and the integrity of subsequent unicast-type time synchronization packets. In order to do this, the server attaches a Message Authentication Code (MAC) to each time synchronization packet. The calculation of the MAC includes the whole time synchronization packet and the cookie which is shared between client and server. Therefore, the client can perform a validity check for this MAC on reception of a time synchronization packet.

4.2. Broadcast Mode

After the client has accomplished the necessary initial time synchronization via client-server mode, the necessary broadcast parameters are communicated from the server to the client. The "broadcast parameter" message exchange described in [I-D.ietf-ntp-network-time-security], Appendix B., can be utilized

for this communication. An implementation MUST support the use of this exchange. It MAY additionally support the use of any alternative secure communication for this purpose, as long as it fulfills the necessary security goals (given in [I-D.ietf-ntp-network-time-security], Section 6.2.1.).

After the client has received the necessary broadcast parameters, "broadcast time synchronization" message exchanges are utilized in combination with optional "broadcast keycheck" exchanges to protect authenticity and integrity of NTP broadcast time synchronization packets. As in the case of unicast time synchronization messages, this is also achieved by MACs.

5. Protocol Sequence

Throughout this section, the nonces, cookies and MACs mentioned have bit lengths of `B_nonce`, `B_cookie` and `B_mac`, respectively. These bit lengths are defined in Appendix B (Appendix B).

5.1. The Client

5.1.1. The Client in Unicast Mode

For a unicast run, the client performs the following steps:

NOTE: Steps 1 through 4 MAY alternatively be replaced an alternative secure mechanism for association and cookie exchange. An implementation MAY choose to replace either steps 1 and 2 or all of the steps 1 through 4 by alternative secure communication.

Step 1: It sends a `client_assoc` message to the server. It MUST keep the transmitted nonce as well as the values for the version number and algorithms available for later checks.

Step 2: It waits for a reply in the form of a `server_assoc` message. After receipt of the message it performs the following checks:

- * The client checks that the message contains a conforming version number.
- * It checks that the nonce sent back by the server matches the one transmitted in `client_assoc`,
- * It also verifies that the server has chosen the encryption and hash algorithms from its proposal sent in the `client_assoc` message and that this proposal was not altered.

- * Furthermore, it performs authenticity checks on the certificate chain and the signature.

If one of the checks fails, the client MUST abort the run.

Discussion: Note that by performing the above message exchange and checks, the client validates the authenticity of its immediate NTP server only. It does not recursively validate the authenticity of each NTP server on the time synchronization chain. Recursive authentication (and authorization) as formulated in RFC 7384 [RFC7384] depends on the chosen trust anchor.

Step 3: Next it sends a `client_cook` message to the server. The client MUST save the included nonce until the reply has been processed.

Step 4: It awaits a reply in the form of a `server_cook` message; upon receipt it executes the following actions:

- * It verifies that the received version number matches the one negotiated beforehand.
- * It verifies the signature using the server's public key. The signature has to authenticate the encrypted data.
- * It decrypts the encrypted data with its own private key.
- * It checks that the decrypted message is of the expected format: the concatenation of a `B_nonce` bit nonce and a `B_cookie` bit cookie.
- * It verifies that the received nonce matches the nonce sent in the `client_cook` message.

If one of those checks fails, the client MUST abort the run.

Step 5: The client sends a `time_request` message to the server. The client MUST save the included nonce and the `transmit_timestamp` (from the time synchronization data) as a correlated pair for later verification steps.

Step 6: It awaits a reply in the form of a `time_response` message. Upon receipt, it checks:

- * that the transmitted version number matches the one negotiated previously,

- * that the transmitted nonce belongs to a previous `time_request` message,
- * that the `transmit_timestamp` in that `time_request` message matches the corresponding time stamp from the synchronization data received in the `time_response`, and
- * that the appended MAC verifies the received synchronization data, version number and nonce.

If at least one of the first three checks fails (i.e. if the version number does not match, if the client has never used the nonce transmitted in the `time_response` message, or if it has used the nonce with initial time synchronization data different from that in the response), then the client **MUST** ignore this `time_response` message. If the MAC is invalid, the client **MUST** do one of the following: abort the run or go back to step 3 (because the cookie might have changed due to a server seed refresh). If both checks are successful, the client **SHOULD** continue time synchronization by repeating the exchange of `time_request` and `time_response` messages.

The client's behavior in unicast mode is also expressed in Figure 1.

5.1.2. The Client in Broadcast Mode

To establish a secure broadcast association with a broadcast server, the client **MUST** initially authenticate the broadcast server and securely synchronize its time with it up to an upper bound for its time offset in unicast mode. After that, the client performs the following steps:

NOTE: Steps 1 and 2 **MAY** be replaced by an alternative security mechanism for the broadcast parameter exchange.

Step 1: It sends a `client_bpar` message to the server. It **MUST** remember the transmitted values for the nonce, the version number and the signature algorithm.

Step 2: It waits for a reply in the form of a `server_bpar` message after which it performs the following checks:

- * The message must contain all the necessary information for the TESLA protocol, as specified for a `server_bpar` message.
- * The message must contain a nonce belonging to a `client_bpar` message that the client has previously sent.

- * Verification of the message's signature.

If any information is missing or if the server's signature cannot be verified, the client MUST abort the broadcast run. If all checks are successful, the client MUST remember all the broadcast parameters received for later checks.

Step 3: The client awaits time synchronization data in the form of a server_broadcast message. Upon receipt, it performs the following checks:

1. Proof that the MAC is based on a key that is not yet disclosed (packet timeliness). This is achieved via a combination of checks. First, the disclosure schedule is used, which requires loose time synchronization. If this is successful, the client obtains a stronger guarantee via a key check exchange: it sends a client_keycheck message and waits for the appropriate response. Note that it needs to memorize the nonce and the time interval number that it sends as a correlated pair. For more detail on both of the mentioned timeliness checks, see [I-D.ietf-ntp-network-time-security]. If its timeliness is verified, the packet will be buffered for later authentication. Otherwise, the client MUST discard it. Note that the time information included in the packet will not be used for synchronization until its authenticity could also be verified.
2. The client checks that it does not already know the disclosed key. Otherwise, the client SHOULD discard the packet to avoid a buffer overrun. If verified, the client ensures that the disclosed key belongs to the one-way key chain by applying the one-way function until equality with a previous disclosed key is shown. If it is falsified, the client MUST discard the packet.
3. If the disclosed key is legitimate, then the client verifies the authenticity of any packet that it has received during the corresponding time interval. If authenticity of a packet is verified it is released from the buffer and the packet's time information can be utilized. If the verification fails, then authenticity is no longer given. In this case, the client MUST request authentic time from the server by means of a unicast time request message. Also, the client MUST re-initialize the broadcast sequence with a "client_bpar" message if the one-way key chain expires, which it can check via the disclosure schedule.

See RFC 4082 [RFC4082] for a detailed description of the packet verification process.

The client MUST restart the broadcast sequence with a client_bpar message ([I-D.ietf-ntp-network-time-security]) if the one-way key chain expires.

The client's behavior in broadcast mode can also be seen in Figure 2.

5.2. The Server

5.2.1. The Server in Unicast Mode

To support unicast mode, the server MUST be ready to perform the following actions:

- o Upon receipt of a client_assoc message, the server constructs and sends a reply in the form of a server_assoc message as described in [I-D.ietf-ntp-network-time-security].
- o Upon receipt of a client_cook message, the server checks whether it supports the given cryptographic algorithms. It then calculates the cookie according to the formula given in [I-D.ietf-ntp-network-time-security]. With this, it MUST construct a server_cook message as described in [I-D.ietf-ntp-network-time-security].
- o Upon receipt of a time_request message, the server re-calculates the cookie, then computes the necessary time synchronization data and constructs a time_response message as given in [I-D.ietf-ntp-network-time-security].

The server MUST refresh its server seed periodically (see [I-D.ietf-ntp-network-time-security]).

In addition to the server MAY be ready to perform the following action:

- o If an external mechanism for association and key exchange is used the server has to react accordingly.

5.2.2. The Server in Broadcast Mode

A broadcast server MUST also support unicast mode in order to provide the initial time synchronization, which is a precondition for any broadcast association. To support NTS broadcast, the server MUST additionally be ready to perform the following actions:

- o Upon receipt of a `client_bpar` message, the server constructs and sends a `server_bpar` message as described in [I-D.ietf-ntp-network-time-security].
- o Upon receipt of a `client_keycheck` message, the server looks up whether it has already disclosed the key associated with the interval number transmitted in that message. If it has not disclosed it, it constructs and sends the appropriate `server_keycheck` message as described in [I-D.ietf-ntp-network-time-security]. For more details, see also [I-D.ietf-ntp-network-time-security].
- o The server follows the TESLA protocol in all other aspects, by regularly sending `server_broad` messages as described in [I-D.ietf-ntp-network-time-security], adhering to its own disclosure schedule.

The server is responsible to watch for the expiration date of the one-way key chain and generate a new key chain accordingly.

In addition to the items above, the server MAY be ready to perform the following action:

- o Upon receipt of external communication for the purpose of broadcast parameter exchange, the server reacts according to the way the external communication is specified.

6. Implementation Notes: ASN.1 Structures and Use of the CMS

This section presents some hints about the structures of the communication packets for the different message types when one wishes to implement NTS for NTP. See document [I-D.ietf-ntp-cms-for-nts-message] for descriptions of the archetypes for CMS structures as well as for the ASN.1 structures that are referenced here.

6.1. Unicast Messages

6.1.1. Association Messages

6.1.1.1. Message Type: "client_assoc"

This message is realized as an NTP packet with an extension field which holds an "NTS-Plain" archetype structure. This structure consists only of an NTS message object of the type "ClientAssociationData", which holds all the data necessary for the NTS security mechanisms.

6.1.1.2. Message Type: "server_assoc"

Like "client_assoc", this message is realized as an NTP packet with an extension field which holds an "NTS-Plain" archetype structure, i.e. just an NTS message object of the type "ServerAssociationData". The latter holds all the data necessary for NTS.

6.1.2. Cookie Messages

6.1.2.1. Message Type: "client_cook"

This message type is realized as an NTP packet with an extension field which holds a CMS structure of archetype "NTS-Certified", containing in its core an NTS message object of the type "ClientCookieData". The latter holds all the data necessary for the NTS security mechanisms.

6.1.2.2. Message Type: "server_cook"

This message type is realized as an NTP packet with an extension field containing a CMS structure of archetype "NTS-Encrypted-and-Signed". The NTS message object in that structure is a "ServerCookieData" object, which holds all data required by NTS for this message type.

6.1.3. Time Synchronization Messages

6.1.3.1. Message Type: "time_request"

This message type is realized as an NTP packet which actually contains regular NTP time synchronization data, as an unsecured NTP packet from a client to a server would. Furthermore, the packet has an extension field which contains an ASN.1 object of type "TimeRequestSecurityData" (packed in a CMS structure of archetype "NTS-Plain").

6.1.3.2. Message Type: "time_response"

This message is also realized as an NTP packet with regular NTP time synchronization data. The packet also has an extension field which contains an ASN.1 object of type "TimeResponseSecurityData". Finally, this NTP packet has a MAC field which contains a Message Authentication Code generated over the whole packet (including the extension field).

6.2. Broadcast Messages

6.2.1. Broadcast Parameter Messages

6.2.1.1. Message Type: "client_bpar"

This first broadcast message is realized as an NTP packet which is empty except for an extension field which contains an ASN.1 object of type "BroadcastParameterRequest" (packed in a CMS structure of archetype "CMS-Plain"). This is sufficient to transport all data specified by NTS.

6.2.1.2. Message Type: "server_bpar"

This message type is realized as an NTP packet whose extension field carries the necessary CMS structure (archetype: "NTS-Signed"). The NTS message object in this case is an ASN.1 object of type "BroadcastParameterResponse".

6.2.2. Broadcast Time Synchronization Message

6.2.2.1. Message Type: "server_broad"

This message's realization works via an NTP packet which carries regular NTP broadcast time data as well as an extension field, which contains an ASN.1 object of type "BroadcastTime" (packed in a CMS structure with archetype "NTS-Plain"). In addition to all this, this packet has a MAC field which contains a Message Authentication Code generated over the whole packet (including the extension field).

6.2.3. Broadcast Keycheck

6.2.3.1. Message Type: "client_keycheck"

This message is realized as an NTP packet with an extension field, which transports a CMS structure of archetype "NTS-Plain", containing an ASN.1 object of type "ClientKeyCheckSecurityData".

6.2.3.2. Message Type: "server_keycheck"

This message is also realized as an NTP packet with an extension field, which contains an ASN.1 object of type "ServerKeyCheckSecurityData" (packed in a CMS structure of archetype "NTS-Plain"). Additionally, this NTP packet has a MAC field which contains a Message Authentication Code generated over the whole packet (including the extension field).

7. Security Considerations

7.1. Employing Alternative Means for Association and Cookie Exchange

If an implementation uses alternative means to perform association and cookie exchange, it **MUST** make sure that an adversary cannot abuse the server to obtain a cookie belonging to a chosen KIV.

7.2. Usage of NTP Pools

The certification-based authentication scheme described in [I-D.ietf-ntp-network-time-security] is not applicable to the concept of NTP pools. Therefore, NTS is unable to provide secure usage of NTP pools.

7.3. Server Seed Lifetime

tbd

7.4. Supported Hash Algorithms

The list of the hash algorithms supported by the server has to fulfill the following requirements:

- o it **MUST NOT** include SHA-1 or weaker algorithms,
- o it **MUST** include SHA-256 or stronger algorithms.

8. Acknowledgements

The authors would like to thank Russ Housley, Steven Bellovin, David Mills and Kurt Roeckx for discussions and comments on the design of NTS. Also, thanks to Harlan Stenn for his technical review and specific text contributions to this document.

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4082] Perrig, A., Song, D., Canetti, R., Tygar, J., and B. Briscoe, "Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction", RFC 4082, June 2005.

- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, September 2009.
- [RFC5905] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

9.2. Informative References

- [I-D.ietf-ntp-cms-for-nts-message]
Sibold, D., Teichel, K., Roettger, S., and R. Housley,
"Protecting Network Time Security Messages with the
Cryptographic Message Syntax (CMS)", draft-ietf-ntp-cms-
for-nts-message-03 (work in progress), April 2015.
- [I-D.ietf-ntp-network-time-security]
Sibold, D., Roettger, S., and K. Teichel, "Network Time
Security", draft-ietf-ntp-network-time-security-08 (work
in progress), March 2015.
- [RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in
Packet Switched Networks", RFC 7384, October 2014.

Appendix A. Flow Diagrams of Client Behaviour

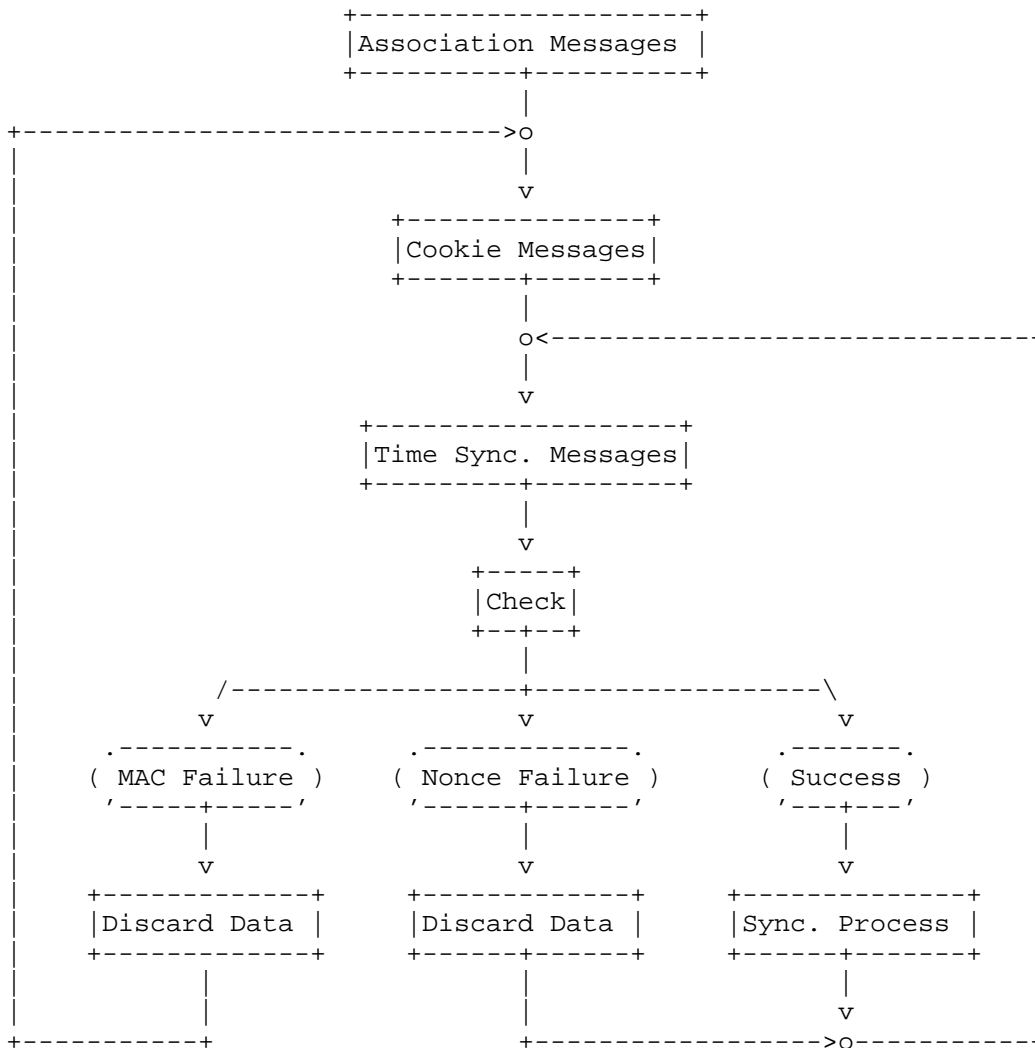


Figure 1: The client's behavior in NTS unicast mode.

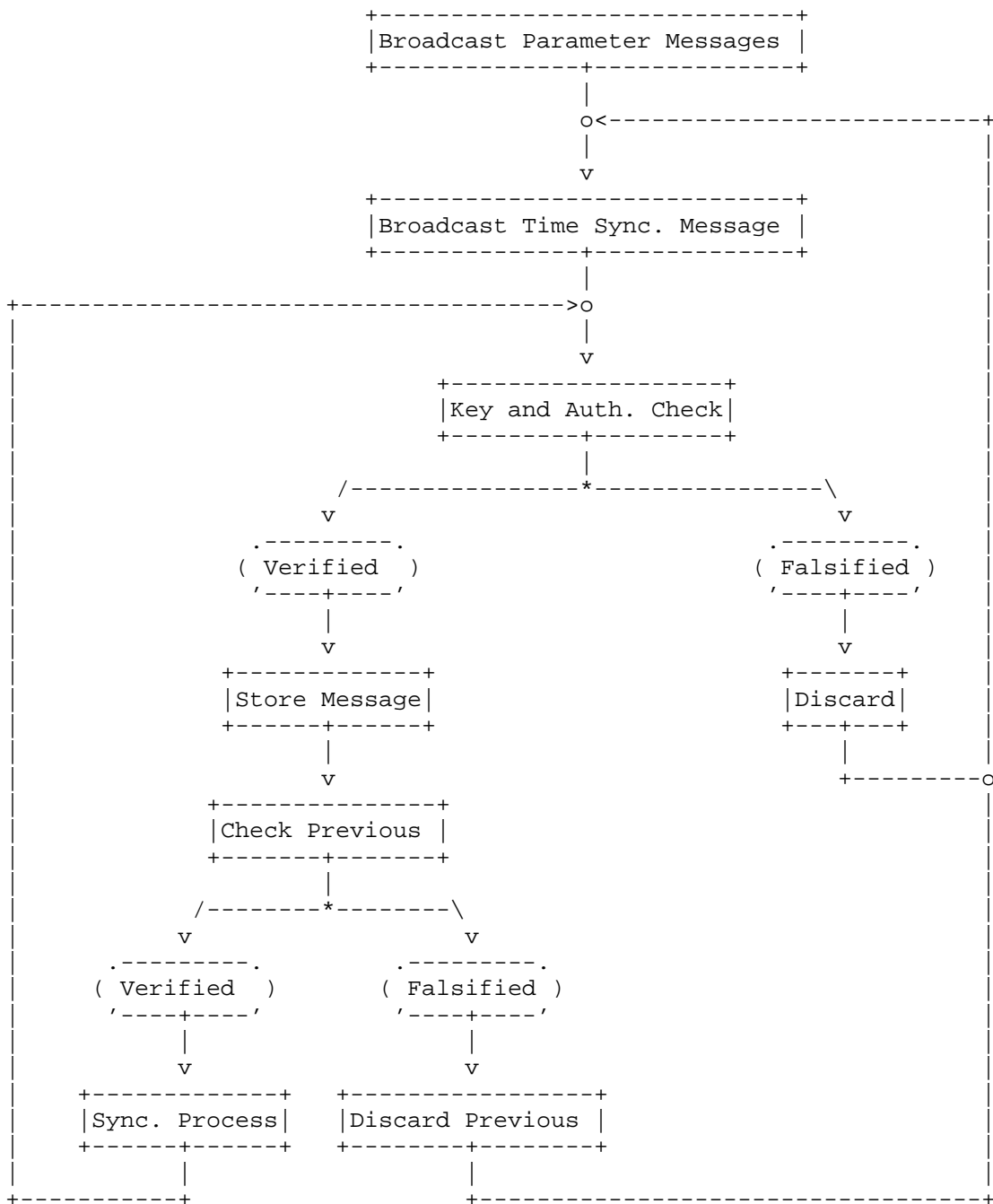


Figure 2: The client's behaviour in NTS broadcast mode.

Appendix B. Bit Lengths for Employed Primitives

Define the following bit lengths for nonces, cookies and MACs:

B_nonce = 128,

B_cookie = 128, and

B_mac = 128.

Authors' Addresses

Dieter Sibold
Physikalisch-Technische Bundesanstalt
Bundesallee 100
Braunschweig D-38116
Germany

Phone: +49-(0)531-592-8420
Fax: +49-531-592-698420
Email: dieter.sibold@ptb.de

Stephen Roettger
Google Inc

Email: stephen.roettger@googlemail.com

Kristof Teichel
Physikalisch-Technische Bundesanstalt
Bundesallee 100
Braunschweig D-38116
Germany

Phone: +49-(0)531-592-8421
Email: kristof.teichel@ptb.de

INTERNET-DRAFT

Enterprise Profile for PTP

July 2018

TICTOC Working Group
Internet Draft
Intended status: Standards Track

Doug Arnold
Meinberg-USA
Heiko Gerstung
Meinberg

Expires: January 31, 2019

Enterprise Profile for the Precision Time Protocol
With Mixed Multicast and Unicast Messages

draft-ietf-tictoc-ntp-enterprise-profile-11.txt

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, except to publish it as an RFC and to translate it into languages other than English.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>

This Internet-Draft will expire on January 31, 2019.

Copyright Notice

Copyright (c) 2018 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 (<http://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.

Abstract

This document describes a profile for the use of the Precision Time Protocol in an IPV4 or IPv6 Enterprise information system environment. The profile uses the End to End Delay Measurement Mechanism, allows both multicast and unicast Delay Request and Delay Response Messages.

Table of Contents

1.	Introduction	2
2.	Conventions used in this document	3
3.	Technical Terms	3
4.	Problem Statement	5
5.	Network Technology	6
6.	Time Transfer and Delay Measurement	7
7.	Default Message Rates	8
8.	Requirements for Master Clocks	8
9.	Requirements for Slave Clocks	8
10.	Requirements for Transparent Clocks	9
11.	Requirements for Boundary Clocks	9
12.	Management and Signaling Messages	9
13.	Forbidden PTP Options	9
14.	Interoperation with Other PTP Profiles	10
15.	Profile Identification	10
16.	Security Considerations	10
17.	IANA Considerations	10
18.	References	11
	18.1. Normative References	11
	18.2. Informative References	11
19.	Acknowledgments	11
20.	Authors addresses	12

1. Introduction

The Precision Time Protocol ("PTP"), standardized in IEEE 1588, has been designed in its first version (IEEE 1588-2002) with the goal to minimize configuration on the participating nodes. Network communication was based solely on multicast messages, which unlike NTP did not require that a receiving node ("slave clock") in [IEEE1588] needs to know the identity of the time sources in the network (the Master Clocks).

The "Best Master Clock Algorithm" ([IEEE1588] Subclause 9.3), a mechanism that all participating PTP nodes must follow, set up strict rules for all members of a PTP domain to determine which node shall be the active sending time source (Master Clock). Although the multicast communication model has advantages in smaller networks, it complicated the application of PTP in larger networks, for example in environments like IP based telecommunication networks or financial data centers. It is considered inefficient that, even if the content of a message applies only to one receiver, it is forwarded by the underlying

network (IP) to all nodes, requiring them to spend network bandwidth and other resources, such as CPU cycles, to drop the message.

The second revision of the standard (IEEE 1588-2008) is the current version (also known as PTPv2) and introduced the possibility to use unicast communication between the PTP nodes in order to overcome the limitation of using multicast messages for the bi-directional information exchange between PTP nodes. The unicast approach avoided that, in PTP domains with a lot of nodes, devices had to throw away more than 99% of the received multicast messages because they carried information for some other node. PTPv2 also introduced PTP profiles ([IEEE1588] subclause 19.3).

This construct allows organizations to specify selections of attribute values and optional features, simplifying the configuration of PTP nodes for a specific application. Instead of having to go through all possible parameters and configuration options and individually set them up, selecting a profile on a PTP node will set all the parameters that are specified in the profile to a defined value. If a PTP profile definition allows multiple values for a parameter, selection of the profile will set the profile-specific default value for this parameter. Parameters not allowing multiple values are set to the value defined in the PTP profile. Many PTP features and functions are optional, and a profile should also define which optional features of PTP are required, permitted, or prohibited. It is possible to extend the PTP standard with a PTP profile by using the TLV mechanism of PTP (see [IEEE1588] subclause 13.4), defining an optional Best Master Clock Algorithm and a few other ways. PTP has its own management protocol (defined in [IEEE1588] subclause 15.2) but allows a PTP profile specify an alternative management mechanism, for example SNMP.

2. Conventions used in this document

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].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

3. Technical Terms

Acceptable Master Table: A PTP Slave Clock may maintain a list of masters which it is willing to synchronize to.

Alternate Master: A PTP Master Clock, which is not the Best Master, may act as a master with the Alternate Master flag set on the messages it sends.

Announce message: Contains the Master Clock properties of a Master Clock. Used to determine the Best Master.

Best Master: A clock with a port in the master state, operating consistently with the Best Master Clock Algorithm.

Best Master Clock Algorithm: A method for determining which state a port of a PTP clock should be in. The algorithm works by identifying which of several PTP Master capable clocks is the best master. Clocks have priority to become the acting Grandmaster, based on the properties each Master Clock sends in its Announce Message.

Boundary Clock: A device with more than one PTP port. Generally boundary Clocks will have one port in slave state to receive timing and then other ports in master state to re-distribute the timing.

Clock Identity: In IEEE 1588-2008 this is a 64-bit number assigned to each PTP clock which must be unique. Often it is derived from the Ethernet MAC address, since there is already an international infrastructure for assigning unique numbers to each device manufactured.

Domain: Every PTP message contains a domain number. Domains are treated as separate PTP systems in the network. Clocks, however, can combine the timing information derived from multiple domains.

End to End Delay Measurement Mechanism: A network delay measurement mechanism in PTP facilitated by an exchange of messages between a Master Clock and Slave Clock.

Grandmaster: the primary Master Clock within a domain of a PTP system

IEEE 1588: The timing and synchronization standard which defines PTP, and describes the node, system, and communication properties necessary to support PTP.

Master Clock: a clock with at least one port in the master state.

NTP: Network Time Protocol, defined by RFC 5905, see [NTP].

Ordinary Clock: A clock that has a single Precision Time Protocol (PTP) port in a domain and maintains the timescale used in the domain. It may serve as a Master Clock, or be a slave clock.

Peer to Peer Delay Measurement Mechanism: A network delay measurement mechanism in PTP facilitated by an exchange of messages between adjacent devices in a network.

Preferred Master: A device intended to act primarily as the Grandmaster of a PTP system, or as a back up to a Grandmaster.

PTP: The Precision Time Protocol, the timing and synchronization protocol defined by IEEE 1588.

PTP port: An interface of a PTP clock with the network. Note that there may be multiple PTP ports running on one physical interface, for example, a unicast slave which talks to several Grandmaster clocks in parallel.

PTPv2: Refers specifically to the second version of PTP defined by IEEE 1588-2008.

Rogue Master: A clock with a port in the master state, even though it should not be in the master state according to the Best Master Clock Algorithm, and does not set the alternate master flag.

Slave clock: a clock with at least one port in the slave state, and no ports in the master state.

Slave Only Clock: An Ordinary Clock which cannot become a Master Clock.

TLV: Type Length Value, a mechanism for extending messages in networked communications.

Transparent Clock. A device that measures the time taken for a PTP event message to transit the device and then updates the message with a correction for this transit time.

Unicast Discovery: A mechanism for PTP slaves to establish a unicast communication with PTP masters using a configured table of master IP addresses and Unicast Message Negotiation.

Unicast Negotiation: A mechanism in PTP for Slave Clocks to negotiate unicast Sync, announce and Delay Request Message Rates from a Master Clock.

4. Problem Statement

This document describes a version of PTP intended to work in large enterprise networks. Such networks are deployed, for example, in financial corporations. It is becoming increasingly common in such networks to perform distributed time tagged measurements, such as one-way packet latencies and cumulative delays on software systems spread across multiple computers. Furthermore, there is often a desire to check the age of information time tagged by a different machine. To perform these measurements, it is necessary to deliver a common precise time to multiple devices on a network. Accuracy currently required in the Financial Industry range from 100 microseconds to 100 nanoseconds to the Grandmaster. This profile does not specify timing performance requirements, but such requirements explain why the needs cannot always be met by NTP, as commonly implemented. Such accuracy cannot usually be achieved with a traditional time transfer such as NTP, without adding

non-standard customizations such as hardware time stamping, and on path support. These features are currently part of PTP, or are allowed by it. Because PTP has a complex range of features and options it is necessary to create a profile for enterprise networks to achieve interoperability between equipment manufactured by different vendors.

Although enterprise networks can be large, it is becoming increasingly common to deploy multicast protocols, even across multiple subnets. For this reason, it is desired to make use of multicast whenever the information going to many destinations is the same. It is also advantageous to send information which is unique to one device as a unicast message. The latter can be essential as the number of PTP slaves becomes hundreds or thousands.

PTP devices operating in these networks need to be robust. This includes the ability to ignore PTP messages which can be identified as improper, and to have redundant sources of time.

Interoperability among independent implementations of this PTP profile has been demonstrated at the ISPCS Plugfest [ISPCS].

5. Network Technology

This PTP profile SHALL operate only in networks characterized by UDP [RFC768] over either IPv4 [RFC791] or IPv6 [RFC2460], as described by Annexes D and E in [IEEE1588] respectively. If a network contains both IPv4 and IPv6, then they SHALL be treated as separate communication paths. Clocks which communicate using IPv4 can interact with clocks using IPv6 if there is an intermediary device which simultaneously communicates with both IP versions. A Boundary Clock might perform this function, for example. A PTP domain SHALL use either IPv4 or IPv6 over a communication path, but not both. The PTP system MAY include switches and routers. These devices MAY be Transparent Clocks, boundary Clocks, or neither, in any combination. PTP Clocks MAY be Preferred Masters, Ordinary Clocks, or Boundary Clocks. The Ordinary Clocks may be Slave Only Clocks, or be master capable.

Note that clocks SHOULD always be identified by their clock ID and not the IP or Layer 2 address. This is important in IPv6 networks since Transparent Clocks are required to change the source address of any packet which they alter. In IPv4 networks some clocks might be hidden behind a NAT, which hides their IP addresses from the rest of the network. Note also that the use of NATs may place limitations on the topology of PTP networks, depending on the port forwarding scheme employed. Details of implementing PTP with NATs are out of scope of this document.

PTP, like NTP, assumes that the one-way network delay for Sync Messages and Delay Response Messages are the same. When this is not true it can cause errors in the transfer of time from the Master to the Slave. It is up to the system integrator to design the network so that such effects do not prevent the PTP system from meeting the timing requirements. The details of

network asymmetry are outside the scope of this document. See for example, [G8271].

6. Time Transfer and Delay Measurement

Master Clocks, Transparent Clocks and Boundary Clocks MAY be either one-step clocks or two-step clocks. Slave clocks MUST support both behaviors. The End to End Delay Measurement Method MUST be used.

Note that, in IP networks, Sync messages and Delay Request messages exchanged between a master and slave do not necessarily traverse the same physical path. Thus, wherever possible, the network SHOULD be traffic engineered so that the forward and reverse routes traverse the same physical path. Traffic engineering techniques for path consistency are out of scope of this document.

Sync messages MUST be sent as PTP event multicast messages (UDP port 319) to the PTP primary IP address. Two step clocks SHALL send Follow-up messages as PTP general messages (UDP port 320). Announce messages MUST be sent as multicast messages (UDP port 320) to the PTP primary address. The PTP primary IP address is 224.0.1.129 for IPv4 and FF0X:0:0:0:0:0:0:181 for Ipv6, where X can be a value between 0x0 and 0xF, see [IEEE1588] Annex E, Section E.3.

Delay Request Messages MAY be sent as either multicast or unicast PTP event messages. Master Clocks SHALL respond to multicast Delay Request messages with multicast Delay Response PTP general messages. Master Clocks SHALL respond to unicast Delay Request PTP event messages with unicast Delay Response PTP general messages. This allow for the use of Ordinary Clocks which do not support the Enterprise Profile, if they are slave Only Clocks.

Clocks SHOULD include support for multiple domains. The purpose is to support multiple simultaneous masters for redundancy. Leaf devices (non-forwarding devices) can use timing information from multiple masters by combining information from multiple instantiations of a PTP stack, each operating in a different domain. Redundant sources of timing can be ensembled, and/or compared to check for faulty Master Clocks. The use of multiple simultaneous masters will help mitigate faulty masters reporting as healthy, network delay asymmetry, and security problems. Security problems include man-in-the-middle attacks such as delay attacks, packet interception / manipulation attacks. Assuming the path to each master is different, failures malicious or otherwise would have to happen at more than one path simultaneously. Whenever feasible, the underlying network transport technology SHOULD be configured so that timing messages in different domains traverse different network paths.

7. Default Message Rates

The Sync, Announce and Delay Request default message rates SHALL each be once per second. The Sync and Delay Request message rates MAY be set to other values, but not less than once every 128 seconds, and not more than 128 messages per second. The Announce message rate SHALL NOT be changed from the default value. The Announce Receipt Timeout Interval SHALL be three Announce Intervals for Preferred Masters, and four Announce Intervals for all other masters.

The logMessageInterval carried in the unicast Delay Response message MAY be set to correspond to the master ports preferred message period, rather than 7F, which indicates message periods are to be negotiated. Note that negotiated message periods are not allowed, see section 13.

8. Requirements for Master Clocks

Master Clocks SHALL obey the standard Best Master Clock Algorithm from [IEEE1588]. PTP systems using this profile MAY support multiple simultaneous Grandmasters if each active Grandmaster is operating in a different PTP domain.

A port of a clock SHALL NOT be in the master state unless the clock has a current value for the number of UTC leap seconds.

If a unicast negotiation signaling message is received it SHALL be ignored.

9. Requirements for Slave Clocks

Slave clocks MUST be able to operate properly in a network which contains multiple Masters in multiple domains. Slaves SHOULD make use of information from the all Masters in their clock control subsystems. Slave Clocks MUST be able to operate properly in the presence of a Rogue Master. Slaves SHOULD NOT Synchronize to a Master which is not the Best Master in its domain. Slaves will continue to recognize a Best Master for the duration of the Announce Time Out Interval. Slaves MAY use an Acceptable Master Table. If a Master is not an Acceptable Master, then the Slave MUST NOT synchronize to it. Note that IEEE 1588-2008 requires slave clocks to support both two-step or one-step Master clocks. See [IEEE1588], subClause 11.2.

Since Announce messages are sent as multicast messages slaves can obtain the IP addresses of a master from the Announce messages.

Note that the IP source addresses of Sync and Follow-up messages may have been replaced by the source addresses of a Transparent Clock, so, slaves MUST send Delay Request messages to the IP address in the Announce message. Sync and Follow-up messages can be correlated with the Announce message using the clock ID, which is never altered by Transparent Clocks in this profile.

10. Requirements for Transparent Clocks

Transparent Clocks SHALL NOT change the transmission mode of an Enterprise Profile PTP message. For example, a Transparent Clock SHALL NOT change a unicast message to a multicast message. Transparent Clocks SHOULD support multiple domains. Transparent Clocks which synchronize to the master clock will need to maintain separate clock rate offsets for each of the supported domains.

11. Requirements for Boundary Clocks

Boundary Clocks SHOULD support multiple simultaneous PTP domains. This will require them to maintain servo loops for each of the domains supported, at least in software. Boundary Clocks MUST NOT combine timing information from different domains.

12. Management and Signaling Messages

PTP Management messages MAY be used. Management messages intended for a specific clock, i.e. the [IEEE1588] defined attribute `targetPortIdentity.clockIdentity` is not set to All 1's, MUST be sent as a unicast message. Similarly, if any signaling messages are used they MUST also be sent as unicast messages whenever the message is intended for a specific clock.

13. Forbidden PTP Options

Clocks operating in the Enterprise Profile SHALL NOT use peer to peer timing for delay measurement. Grandmaster Clusters are NOT ALLOWED. The Alternate Master option is also NOT ALLOWED. Clocks operating in the Enterprise Profile SHALL NOT use Alternate Timescales. Unicast discovery and unicast negotiation SHALL NOT be used.

14. Interoperation with IEEE 1588 Default Profile

Clocks operating in the Enterprise Profile will interoperate with clocks operating in the Default Profile described in [IEEE1588] Annex J.3. This variant of the Default Profile uses the End to End Delay Measurement Mechanism. In addition, the Default Profile would have to operate over IPv4 or IPv6 networks, and use management messages in unicast when those messages are directed at a specific clock. If either of these requirements are not met than Enterprise Profile clocks will not interoperate with Annex J.3 Default Profile Clocks. The Enterprise Profile will not interoperate with the Annex J.4 variant of the Default Profile which requires use of the Peer to Peer Delay Measurement Mechanism.

Enterprise Profile Clocks will interoperate with clocks operating in other profiles if the clocks in the other profiles obey the rules of the Enterprise Profile. These rules MUST NOT be changed to achieve interoperability with other profiles.

15. Profile Identification

The IEEE 1588 standard requires that all profiles provide the following identifying information.

PTP Profile:
Enterprise Profile
Version: 1.0
Profile identifier: 00-00-5E-00-01-00

This profile was specified by the IETF

A copy may be obtained at
<https://datatracker.ietf.org/wg/tictoc/documents>

16. Security Considerations

Protocols used to transfer time, such as PTP and NTP can be important to security mechanisms which use time windows for keys and authorization. Passing time through the networks poses a security risk since time can potentially be manipulated. The use of multiple simultaneous masters, using multiple PTP domains can mitigate problems from rogue masters and man-in-the-middle attacks. See sections 9 and 10. Additional security mechanisms are outside the scope of this document.

PTP native management messages SHOULD not be used, due to the lack of a security mechanism for this option. Secure management can be obtained using standard management mechanisms which include security, for example NETCONF [NETCONF].

General security considerations of time protocols are discussed in [RFC7384].

17. IANA Considerations

There are no IANA requirements in this specification.

18. References

18.1. Normative References

- [IEEE1588] IEEE std. 1588-2008, "IEEE Standard for a Precision Clock Synchronization for Networked Measurement and Control Systems." July, 2008.
- [RFC768] Postel, J., "User Datagram Protocol," RFC 768, August, 1980.
- [RFC791] "Internet Protocol DARPA Internet Program Protocol Specification," RFC 791, September, 1981.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2460] Deering, S., Hinden, R., "Internet Protocol, Version 6 (IPv6) Specification," RFC 2460, December, 1998.

18.2. Informative References

- [G8271] ITU-T G.8271/Y.1366, "Time and Phase Synchronization Aspects of Packet Networks" February, 2012.
- [ISPCS] Arnold, D., et. al. "Plugfest Report," International Symposium on Precision Clock Synchronization for Measurement, Control and Communications, Monterey, CA, October, 2017.
- [NETCONF] Ens, R., et. al., "Network Configuration Protocol (NETCONF)," RFC 6241, June, 2011.
- [NTP] Mills, D., Martin, J., Burbank, J., Kasch, W., "Network Time Protocol Version 4: Protocol and Algorithms Specification," RFC 5905, June 2010.
- [RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks," RFC 7384, October, 2014.

19. Acknowledgments

The authors would like to thank members of IETF for reviewing and providing feedback on this draft.

This document was initially prepared using 2-Word-v2.0.template.dot.

20. Authors' Addresses

Doug Arnold
Meinberg USA
929 Salem End Road
Framingham, MA 01702
USA

Email: doug.arnold@meinberg-usa.com

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

Email: Heiko.gerstung@meinberg.de

Internet Working Group

Y. Jiang

Internet Draft

X. Liu

J. Xu

Huawei

Intended status: Standards Track

Expires: January 2015

July 6, 2015

Yang Data Model for IEEE 1588v2
draft-jlx-tictoc-1588v2-yang-00.txt

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>

This Internet-Draft will expire on January 6, 2013.

Copyright Notice

Copyright (c) 2015 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 (<http://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.

Abstract

This document defines a YANG data model for the configuration of IEEE 1588v2 devices and clocks, and also retrieval of the configuration information, data set and running states of IEEE 1588v2 clocks.

Table of Contents

1.	Introduction	2
2.	Conventions used in this document	3
3.	Terminology	3
4.	IEEE 1588V2 YANG Model	4
4.1.	IEEE 1588V2 YANG model hierarchy	4
4.2.	IEEE 1588v2 rpc definitions	7
5.	IEEE 1588v2 YANG Module	12
6.	Security Considerations	36
7.	IANA Considerations	36
8.	References	36
8.1.	Normative References	36
8.2.	Informative References	36
9.	Acknowledgments	37

1. Introduction

IEEE 1588v2 as a synchronization protocol is widely supported in the carrier networks. It can provide high precision time synchronization as high as nano-seconds. The protocol depends on a PTP engine to automatically decide its state, and a PTP transportation layer to carry the PTP timing and various quality messages. The configuration parameters and state data sets of IEEE 1588v2 are numerous.

Some work on IEEE 1588v2 MIB [PTP-MIB] is in progress in the IETF TICTOC WG. But the work is only scoped with retrieval of the state data of IEEE 1588v2 by SNMP and configuration is not considered, thus its use is limited.

Some service providers require the management of the IEEE 1588v2 synchronization network can be more flexible and more Internet-based (typically overlaid on their transport networks). SDN is another driving factor which demands a greater control over

synchronization networks. Advantages of YANG include Internet based configuration capability, validation, roll-back and etc., all these characteristics make it attractive to become a modeling language for IEEE 1588v2.

YANG [RFC6020] is a data modeling language used to model configuration and state data manipulated by the Network Configuration Protocol (NETCONF) [RFC6241]. A small set of built-in data types are defined in [RFC6020], and a collection of common data types are further defined in [RFC6991].

This document defines a YANG [RFC6020] data model for the configuration of IEEE 1588v2 devices and clocks, and also retrieval of the state data of IEEE 1588v2 clocks.

In order to fulfill the need of a lightweight implementation, the core module is designed to be generic and minimal, but be extensible with capability negotiation. That is, if a node is verified with a capability of more functions, then more modules can be loaded on demand, otherwise, only a basic module is loaded on the node.

This document defines system information, data sets and running states following the structure and definitions in IEEE 1588v2, and in [PTP-MIB].

2. Conventions used in this document

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].

3. Terminology

Terminologies used in this document are extracted from [IEEE1588] and [PTP-MIB].

ARB	Arbitrary Timescale
BC	Boundary Clock
DS	Data Set
E2E	End-to-End
EUI	Extended Unique Identifier.

GPS	Global Positioning System
IANA	Internet Assigned Numbers Authority
IP	Internet Protocol
NIST	National Institute of Standards and Technology
NTP	Network Time Protocol
OC	Ordinary Clock
P2P	Peer-to-Peer
PTP	Precision Time Protocol
TAI	International Atomic Time
TC	Transparent Clock
UDP	User Datagram Protocol
UTC	Coordinated Universal Time

4. IEEE 1588V2 YANG Model

4.1. IEEE 1588V2 YANG model hierarchy

This section describes the hierarchy of IEEE 1588v2 YANG module. Query and retrieval of device wide or port specific configuration information and clock data set is described for this version, while configuration module will be described in a next version.

Query and retrieval of clock information include:

- Device level configuration information, including: clock identity, domain number, clock-running states and statistics.
- Clock data set attributes in a clock node, including: clock-current-DS, clock-parent-DS, clock-default-DS, clock-time-properties-DS, and clock-trans-default-DS and etc.
- Port specific data set attributes, including: clock-port-DS, clock-port-running, clock-port-trans-DS and clock-port-associate and etc.

```

module: ietf-yang-ptp-query
  +--rw device-ptp-query-table
    +--rw ptpDevicequery* [clock-identity domain-number]
      +--rw clock-identity          binary
      +--rw domain-number           uint32
      +--rw clock-current-DS
        +--rw steps-removed?       uint16
        +--rw offset-from-master?  binary
        +--rw mean-path-delay?     binary
      +--rw clock-parent-DS
        +--rw parent-port-identity? binary
        +--rw parent-stats?        boolean
        +--rw offset-scaled-log-variance? int32
        +--rw clock-phase-change-rate? int32
        +--rw GM-clock-identity?    binary
        +--rw GM-clock-priority1?   uint8
        +--rw GM-clock-priority2?   uint8
        +--rw GM-clock-quality-class? uint32
        +--rw GM-clock-quality-accuracy? uint32
        +--rw GM-clock-quality-offset? uint16
      +--rw clock-default-DS
        +--rw two-step-flag?        boolean
        +--rw clock-identity?       binary
        +--rw priority1?            uint8
        +--rw priority2?            uint8
        +--rw slave-only?           boolean
        +--rw quality-class?         uint32
        +--rw quality-accuracy?     uint32
        +--rw quality-offset?       uint16
      +--rw clock-running
        +--rw device-clock-type?    enumeration
        +--rw clock-running-state?  enumeration
        +--rw clock-running-packets-sent? uint64
        +--rw clock-running-packets-received? uint64
      +--rw clock-time-properties-DS
        +--rw current-UTC-offset-valid? boolean
        +--rw current-UTC-offset?    uint16
        +--rw leap59?                 boolean
        +--rw leap61?                 boolean
        +--rw time-traceable?         boolean
        +--rw freq-traceable?         boolean
        +--rw PTP-timescale?          boolean
        +--rw source?                 enumeration
      +--rw clock-trans-default-DS
        +--rw clock-identity?       binary
        +--rw num-of-ports?          uint32
        +--rw delay?                 enumeration

```

```

|           +--rw primary-domain?   uint32
+--rw port-ptp-query-table
  +--rw ptp-port-query* [clock-identity port-number domain-
number]
    +--rw clock-identity             binary
    +--rw port-number                uint32
    +--rw domain-number              uint32
    +--rw clock-port
      | +--rw clock-port-name?       string
      | +--rw port-twostep-flag?    boolean
      | +--rw clock-port-current-peer-address? string
      | +--rw clock-port-num-of-associated-ports? uint32
+--rw clock-port-DS
  | +--rw port-identity
  | | +--rw clock-identity?         binary
  | | +--rw port-number?           uint32
  | +--rw log-announcement-interval? int64
  | +--rw announce-rct-timeout?    uint8
  | +--rw log-sync-interval?       int8
  | +--rw min-delay-req-interval?  uint8
  | +--rw peer-delay-req-interval? uint32
  | +--rw delay-mech?              enumeration
  | +--rw peer-mean-path-delay?    binary
  | +--rw grant-duration?          uint32
  | +--rw PTP-version?             uint32
+--rw clock-port-running
  | +--rw port-type?               enumeration
  | +--rw running-state?           enumeration
  | +--rw running-role?            enumeration
  | +--rw running-interface-index? uint32
  | +--rw running-Tx-mode?         enumeration
  | +--rw running-Rx-mode?         enumeration
  | +--rw running-packets-received? binary
  | +--rw running-packets-sent?    uint64
+--rw clock-port-trans-DS
  | +--rw log-min-pdelay-req-int?  int32
  | +--rw faulty-flag?             boolean
  | +--rw peer-mean-path-delay?    binary
+--rw clock-port-associate
  +--rw port-index?                uint32
  +--rw associate-address?          binary
  +--rw associate-packets-sent?     binary
  +--rw associate-packets-received? binary
  +--rw associate-in-errors?        binary
  +--rw associate-out-errors?       binary

```


4.2. IEEE 1588v2 rpc definitions

The rpc model facilitates issuing commands to a NETCONF server and obtaining a response. Rpc model defined in this document provides commands to retrieve their configuration information, data sets and running states. Configuration of 1588v2 clocks and PTP ports will be provided in a next version.

rpcs:

```

+---x read-device-ptp-query-table
|
| +---w input
| | +---w clock-identity?   binary
| | +---w domain-number?   uint32
| +--ro output
| | +--ro clock-current-DS
| | | +--ro steps-removed?      uint16
| | | +--ro offset-from-master?  binary
| | | +--ro mean-path-delay?    binary
| | +--ro clock-parent-DS
| | | +--ro parent-port-identity?  binary
| | | +--ro parent-stats?          boolean
| | | +--ro offset-scaled-log-variance?  int32
| | | +--ro clock-phase-change-rate?  int32
| | | +--ro GM-clock-identity?      binary
| | | +--ro GM-clock-priority1?    uint8
| | | +--ro GM-clock-priority2?    uint8
| | | +--ro GM-clock-quality-class?  uint32
| | | +--ro GM-clock-quality-accuracy?  uint32
| | | +--ro GM-clock-quality-offset?  uint16
| | +--ro clock-default-DS
| | | +--ro two-step-flag?        boolean
| | | +--ro clock-identity?      binary
| | | +--ro priority1?           uint8
| | | +--ro priority2?           uint8
| | | +--ro slave-only?          boolean
| | | +--ro quality-class?       uint32
| | | +--ro quality-accuracy?    uint32
| | | +--ro quality-offset?     uint16
| | +--ro clock-running
| | | +--ro device-clock-type?      enumeration
| | | +--ro clock-running-state?    enumeration
| | | +--ro clock-running-packets-sent?  uint64
| | | +--ro clock-running-packets-received?  uint64
| | +--ro clock-time-properties-DS
| | | +--ro current-UTC-offset-valid?  boolean
| | | +--ro current-UTC-offset?      uint16
| | | +--ro leap59?                 boolean

```

```

|         |   +--ro leap61?                boolean
|         |   +--ro time-traceable?       boolean
|         |   +--ro freq-traceable?       boolean
|         |   +--ro PTP-timescale?        boolean
|         |   +--ro source?                enumeration
|         |   +--ro clock-trans-default-DS
|         |     +--ro clock-identity?     binary
|         |     +--ro num-of-ports?       uint32
|         |     +--ro delay?               enumeration
|         |     +--ro primary-domain?     uint32
+----x read-clock-current-DS
|   +----w input
|   |   +----w clock-identity?     binary
|   |   +----w domain-number?     uint32
|   +--ro output
|   |   +--ro steps-removed?         uint16
|   |   +--ro offset-from-master?    binary
|   |   +--ro mean-path-delay?       binary
+----x read-clock-parent-DS
|   +----w input
|   |   +----w clock-identity?     binary
|   |   +----w domain-number?     uint32
|   +--ro output
|   |   +--ro parent-port-identity?   binary
|   |   +--ro parent-stats?           boolean
|   |   +--ro offset-scaled-log-variance? int32
|   |   +--ro clock-phase-change-rate? int32
|   |   +--ro GM-clock-identity?      binary
|   |   +--ro GM-clock-priority1?     uint8
|   |   +--ro GM-clock-priority2?     uint8
|   |   +--ro GM-clock-quality-class?  uint32
|   |   +--ro GM-clock-quality-accuracy? uint32
|   |   +--ro GM-clock-quality-offset? uint16
+----x read-clock-default-DS
|   +----w input
|   |   +----w clock-identity?     binary
|   |   +----w domain-number?     uint32
|   +--ro output
|   |   +--ro two-step-flag?          boolean
|   |   +--ro clock-identity?        binary
|   |   +--ro priority1?              uint8
|   |   +--ro priority2?              uint8
|   |   +--ro slave-only?             boolean
|   |   +--ro quality-class?           uint32
|   |   +--ro quality-accuracy?       uint32
|   |   +--ro quality-offset?         uint16
+----x read-clock-running

```

```

| +---w input
| | +---w clock-identity?  binary
| | +---w domain-number?  uint32
+--ro output
  +--ro device-clock-type?          enumeration
  +--ro clock-running-state?        enumeration
  +--ro clock-running-packets-sent?  uint64
  +--ro clock-running-packets-received?  uint64
+---x read-clock-time-properties-DS
| +---w input
| | +---w clock-identity?  binary
| | +---w domain-number?  uint32
+--ro output
  +--ro current-UTC-offset-valid?  boolean
  +--ro current-UTC-offset?        uint16
  +--ro leap59?                    boolean
  +--ro leap61?                    boolean
  +--ro time-traceable?            boolean
  +--ro freq-traceable?            boolean
  +--ro PTP-timescale?            boolean
  +--ro source?                    enumeration
+---x read-clock-trans-default-DS
| +---w input
| | +---w clock-identity?  binary
| | +---w domain-number?  uint32
+--ro output
  +--ro clock-identity?  binary
  +--ro num-of-ports?    uint32
  +--ro delay?           enumeration
  +--ro primary-domain?  uint32
+---x read-port-ptp-query-table
| +---w input
| | +---w clock-identity?  binary
| | +---w port-number?    uint32
| | +---w domain-number?  uint32
+--ro output
  +--ro clock-port
  | +--ro clock-port-name?          string
  | +--ro port-twostep-flag?        boolean
  | +--ro clock-port-current-peer-address?  string
  | +--ro clock-port-num-of-associated-ports?  uint32
  +--ro clock-port-DS
  | +--ro port-identity
  | | +--ro clock-identity?  binary
  | | +--ro port-number?    uint32
  | +--ro log-announcement-interval?  int64
  | +--ro announce-rct-timeout?      uint8

```

```

|         | +--ro log-sync-interval?          int8
|         | +--ro min-delay-req-interval?    uint8
|         | +--ro peer-delay-req-interval?  uint32
|         | +--ro delay-mech?              enumeration
|         | +--ro peer-mean-path-delay?    binary
|         | +--ro grant-duration?         uint32
|         | +--ro PTP-version?            uint32
| +--ro clock-port-running
|         | +--ro port-type?                enumeration
|         | +--ro running-state?          enumeration
|         | +--ro running-role?          enumeration
|         | +--ro running-interface-index? uint32
|         | +--ro running-Tx-mode?       enumeration
|         | +--ro running-Rx-mode?       enumeration
|         | +--ro running-packets-received? binary
|         | +--ro running-packets-sent?   uint64
| +--ro clock-port-trans-DS
|         | +--ro log-min-pdelay-req-int?  int32
|         | +--ro faulty-flag?            boolean
|         | +--ro peer-mean-path-delay?    binary
| +--ro clock-port-associate
|         | +--ro port-index?              uint32
|         | +--ro associate-address?      binary
|         | +--ro associate-packets-sent?  binary
|         | +--ro associate-packets-received? binary
|         | +--ro associate-in-errors?    binary
|         | +--ro associate-out-errors?   binary
+---x read-clock-port
| +---w input
|         | +---w clock-identity?         binary
|         | +---w port-number?           uint32
|         | +---w domain-number?        uint32
| +--ro output
|         | +--ro clock-port-name?        string
|         | +--ro port-twostep-flag?     boolean
|         | +--ro clock-port-current-peer-address? string
|         | +--ro clock-port-num-of-associated-ports? uint32
+---x read-clock-port-DS
| +---w input
|         | +---w clock-identity?         binary
|         | +---w port-number?           uint32
|         | +---w domain-number?        uint32
| +--ro output
|         | +--ro port-identity
|         |         | +--ro clock-identity?  binary
|         |         | +--ro port-number?    uint32
|         | +--ro log-announcement-interval? int64

```

```

|         +--ro announce-rct-timeout?          uint8
|         +--ro log-sync-interval?            int8
|         +--ro min-delay-req-interval?       uint8
|         +--ro peer-delay-req-interval?      uint32
|         +--ro delay-mech?                   enumeration
|         +--ro peer-mean-path-delay?         binary
|         +--ro grant-duration?               uint32
|         +--ro PTP-version?                  uint32
+---x read-clock-port-running
|   +---w input
|   |   +---w clock-identity?    binary
|   |   +---w port-number?      uint32
|   |   +---w domain-number?    uint32
|   +--ro output
|   |   +--ro port-type?          enumeration
|   |   +--ro running-state?      enumeration
|   |   +--ro running-role?      enumeration
|   |   +--ro running-interface-index? uint32
|   |   +--ro running-Tx-mode?    enumeration
|   |   +--ro running-Rx-mode?    enumeration
|   |   +--ro running-packets-received? binary
|   |   +--ro running-packets-sent? uint64
+---x read-clock-port-trans-DS
|   +---w input
|   |   +---w clock-identity?    binary
|   |   +---w port-number?      uint32
|   |   +---w domain-number?    uint32
|   +--ro output
|   |   +--ro log-min-pdelay-req-int? int32
|   |   +--ro faulty-flag?        boolean
|   |   +--ro peer-mean-path-delay? binary
+---x read-clock-port-associate
|   +---w input
|   |   +---w clock-identity?    binary
|   |   +---w port-number?      uint32
|   |   +---w domain-number?    uint32
|   +--ro output
|   |   +--ro port-index?          uint32
|   |   +--ro associate-address?   binary
|   |   +--ro associate-packets-sent? binary
|   |   +--ro associate-packets-received? binary
|   |   +--ro associate-in-errors?  binary
|   |   +--ro associate-out-errors? binary

```

5. IEEE 1588v2 YANG Module

```
module ietf-yang-ptp-query{
  namespace "urn:ietf:params:xml:ns:yang:1588v2";
  prefix "ptp-query";
  organization "IETF TICTOC WG";
  contact "jiangyuanlong@huawei.com";
  description
    "This YANG module defines a data model for the configuration
    of IEEE 1588v2 devices and clocks, and also retrieval of the
    state data of IEEE 1588v2 clocks.";
  revision "2015-07-06"{
  description "Initial revision.";
  reference "draft-jxl-tictoc-1588v2-yang";
  }

  grouping device-identify-infor{
    description
      "This group bundles together all information used to
      identify a device.";
    leaf clock-identity{
      description "identify a specific PTP node.";
      config true;
      type binary {
        length "1..255";
      }
    }

    leaf domain-number{
      description
        "This object specifies the domain number used to create
        logical group of PTP devices.";
      type uint32{
        range "0..255";
      }
    }
  }

  grouping clock-current-DS-entry{
    description
      "This group bundles together all information about the
      PTP clock Current Datasets for a single device.";

    leaf steps-removed{
      description
```

```
        "specifies the distance measured by the number of
        Boundary clocks between the local clock and the
        Foreign master as indicated in the stepsRemoved
        field of Announce messages.";
    type uint16;
}
leaf offset-from-master{
    description
        "specifies the current clock dataset ClockOffset value.
        The value of the computation of the offset in time
        between a slave and a master clock.";
    type binary {
        length "1..255";
    }
}
leaf mean-path-delay{
    description
        "The mean path delay between a pair of ports as measure
        by the delay request-response mechanism.";
    type binary {
        length "1..255";
    }
}
}

grouping clock-parent-DS-entry{
    description
        "This group bundles together all information about the PTP
        clock Parent Datasets for a single device.";

    leaf parent-port-identity{
        description
            "specifies the value of portIdentity of the port on the
            master that issues the Sync messages used in
            synchronizing this clock.";
        type binary{
            length "1..256";
        }
    }
    leaf parent-stats{
        description
            "specifies the Parent Dataset ParentStats value. This
            value indicates whether the values of ParentDSOffset and
            ParentDSClockPhChRate have been measured and are valid.
            A TRUE value shall indicate valid data";
        type boolean;
    }
}
```

```
leaf offset-scaled-log-variance{
  description
    "specifies the Parent Dataset
    ParentOffsetScaledLogVariance value. This value is the
    variance of the parent clocks phase as measured by the
    local clock.";
  type int32{
    range "-128..127";
  }
}
leaf clock-phase-change-rate{
  description
    "specifies the clock's parent dataset
    ParentClockPhaseChangeRate value. This value is an
    estimate of the parent clocks phase change rate as
    measured by the slave clock.";
  type int32;
}
leaf GM-clock-identity{
  description
    "This object specifies the parent dataset Grandmaster
    clock identity";
  type binary{
    length "1..255";
  }
}
leaf GM-clock-priority1{
  description
    "This object specifies the parent dataset Grandmaster
    clock priority1.";
  type uint8;
}
leaf GM-clock-priority2{
  description
    "This object specifies the parent dataset grandmaster
    clock priority2.";
  type uint8;
}
leaf GM-clock-quality-class{
  description
    "This object specifies the parent dataset grandmaster
    clock quality class.";
  type uint32{
    range "0..255";
  }
}
leaf GM-clock-quality-accuracy{
```



```
        description
            "This object specifies the parent dataset grandmaster
            clock quality accuracy";
        type uint32;
    }
    leaf GM-clock-quality-offset{
        description
            "This object specifies the parent dataset grandmaster
            clock quality offset.";
        type uint16;
    }
}

grouping clock-default-DS-entry{
    description
        "This group bundles together all information about the
        PTP clock Default Datasets for a single device.";

    leaf two-step-flag{
        description
            "This object specifies whether the Two Step process is
            used.";
        type boolean;
    }
    leaf clock-identity{
        description
            "This object specifies the default Datasets clock
            identity";
        type binary{
            length "1..255";
        }
    }
    leaf priority1{
        description
            "This object specifies the default Datasets clock
            Priority1";
        type uint8;
    }
    leaf priority2{
        description
            "This object specifies the default Datasets clock
            Priority2";
        type uint8;
    }
    leaf slave-only{
        description "Whether the SlaveOnly flag is set";
        type boolean;
    }
}
```

```
    }
    leaf quality-class{
      description
        "This object specifies the default dataset Quality
        Class.";
      type uint32{
        range "0..255";
      }
    }
    leaf quality-accuracy{
      description
        "This object specifies the default dataset Quality
        Accuracy.";
      type uint32;
    }
    leaf quality-offset{
      description
        "This object specifies the default dataset Quality
        offset.";
      type uint16;
    }
  }
}

grouping clock-running-entry{
  description
    "This group bundles together all information about the
    PTP clock Running Datasets for a single device.";

  leaf device-clock-type{
    description "This object specifies the clock type.";
    type enumeration{
      enum oc{
        value 0;
        description "ordinary clock ";
      }
      enum bc{
        value 1;
        description "boundary clock";
      }
      enum e2etc{
        value 2;
        description "End-to-end transparent clock";
      }
      enum p2ptc{
        value 3;
        description "Peer-to-peer transparent clock ";
      }
    }
  }
}
```

```
enum e2etcoc{
  value 4;
  description
    "End-to-End transparent clock ordinary clock";
}
enum p2ptcoc{
  value 5;
  description
    "peer-to-peer transparent clock ordinary clock ";
}
enum tcandbc{
  value 6;
  description "transparent clock and boundary clock";
}
}

leaf clock-running-state{
  description
    "specifies the Clock state returned by PTP engine which
    was described earlier.";
  type enumeration{
    enum freerun{
      value 1;
      description
        "Applies to a slave device that is not locked to a
        master. This is the initial state a slave starts
        out with when it
        is not getting any PTP packets from the master or
        because of some other input error (erroneous
        packets, etc).";
    }
    enum holdover{
      value 2;
      description
        "In this state the slave device is locked to a
        master but communication with the master has been
        lost or the timestamps in the ptp packets are
        incorrect. Since the slave was locked to the
        master, it can run in this state, with similar
        accuracy for some time. If communication with the
        master is not restored for an extended period
        (dependent on the clock implementation), the
        device should move to the FREERUN state.";
    }
    enum acquiring{
      value 3;
    }
  }
}
```

```
        description
            "The slave device is receiving packets from a
            master and is trying to acquire a lock.";
    }
    enum frequencyLocked{
        value 4;
        description
            "Slave device is locked to the Master with respect
            to frequency, but not phase aligned";
    }
    enum phaseAligned{
        value 5;
        description
            "Locked to the master with respect to frequency
            and phase.";
    }
}
}
leaf clock-running-packets-sent{
    description
        "This object specifies the total number of all packet
        Unicast and multicast that have been sent out for this
        clock in this domain for this type.";
    type uint64;
}
leaf clock-running-packets-received{
    description
        "This object specifies the total number of all packet
        Unicast and multicast that have been received for this
        clock in this domain for this type.";
    type uint64;
}
}
grouping clock-time-properties-DS-entry{
    description
        "This group bundles together all information about the
        PTP clock time properties datasets for a single device.";

    leaf current-UTC-offset-valid{
        description
            "This object specifies the timeproperties dataset value
            of whether current UTC offset is valid.";
        type boolean;
    }
    leaf current-UTC-offset{
        description
```

```

    "This object specifies the timeproperties dataset value
    of current UTC offset. In PTP systems whose epoch is the
    PTP epoch, the value of
    timePropertiesDS.currentUtcOffset is the offset between
    TAI and UTC; otherwise the value has no meaning. The
    value shall be in units of seconds. The initialization
    value shall be selected as follows: a) If the
    timePropertiesDS.ptpTimescale is TRUE, the value is the
    value obtained from a primary reference if the value is
    known at the time of initialization, else, b) The value
    shall be the current number of leap seconds (7.2.3)
    when the node is designed.";
    type uint16;
}
leaf leap59{
    description
        "This object specifies the Leap59 value in the clock
        Current Dataset.";
    type boolean;
}
leaf leap61{
    description
        "This object specifies the Leap61 value in the clock
        Current Dataset.";
    type boolean;
}
leaf time-traceable{
    description
        "This object specifies the Timetraceable value in the
        clock Current Dataset.";
    type boolean;
}
leaf freq-traceable{
    description
        "This object specifies the Frequency Traceable value in
        the clock Current Dataset.";
    type boolean;
}
leaf PTP-timescale{
    description
        "This object specifies the PTP Timescale value in the
        clock Current Dataset.";
    type boolean;
}
leaf source{
    description

```

```
        "This object specifies the Timesource value in the clock
        Current Dataset.";
    type enumeration{
        enum atomicClock{
            value 16;
        }
        enum gps{
            value 32;
        }
        enum terrestrialRadio{
            value 48;
        }
        enum ptp{
            value 64;
        }
        enum ntp{
            value 80;
        }
        enum handSet{
            value 96;
        }
        enum other{
            value 144;
        }
        enum internalOscillator{
            value 160;
        }
    }
}

grouping clock-trans-default-DS-entry{
    description
        "This group bundles together all information about the
        PTP Transparent clock Default Datasets for a single
        device.";

    leaf clock-identity{
        description
            "This object specifies the value of the clockIdentity
            attribute of the local clock.";
        type binary{
            length "0..255";
        }
    }
    leaf num-of-ports{
        description

```

```
        "This object specifies the number of PTP ports of the
        device.";
    type uint32;
}
leaf delay{
    description
        "This object, if the transparent clock is an end-to-end
        transparent clock, has the value shall be E2E; If the
        transparent clock is a peer-to-peer transparent clock,
        the value shall be P2P.";
    type enumeration{
        enum E2E{
            value 1;
            description
                "The port is configured to use the delay request-
                response mechanism.";
        }
        enum P2P{
            value 2;
            description
                "The port is configured to use the peer delay
                mechanism.";
        }
        enum DISABLED{
            value 254;
            description
                "The port does not implement the delay mechanism.";
        }
    }
}
leaf primary-domain{
    description
        "This object specifies the value of the primary
        syntonization domain. The initialization value shall be
        0.";
    type uint32{
        range "0..255";
    }
}
}

container device-ptp-query-table{
    list ptpDevicequery{
        description "";

        key "clock-identity domain-number";
        uses device-identify-infor;
    }
}
```

```
    container clock-current-DS{
      uses clock-current-DS-entry;
    }
    container clock-parent-DS{
      uses clock-parent-DS-entry;
    }
    container clock-default-DS{
      uses clock-default-DS-entry;
    }
    container clock-running{
      uses clock-running-entry;
    }
    container clock-time-properties-DS{
      uses clock-time-properties-DS-entry;
    }
    container clock-trans-default-DS{
      uses clock-trans-default-DS-entry;
    }
  }
}

rpc read-device-ptp-query-table{
  description
    "Read all device-level PTP informations of a specific
    device.";
  input{
    uses device-identify-infor;
  }
  output{
    container clock-current-DS{
      uses clock-current-DS-entry;
    }
    container clock-parent-DS{
      uses clock-parent-DS-entry;
    }
    container clock-default-DS{
      uses clock-default-DS-entry;
    }
    container clock-running{
      uses clock-running-entry;
    }
    container clock-time-properties-DS{
      uses clock-time-properties-DS-entry;
    }
    container clock-trans-default-DS{
      uses clock-trans-default-DS-entry;
    }
  }
}
```



```
    }  
  }  
  rpc read-clock-current-DS {  
    description  
      "Reads the information about the PTP clock Current  
      Datasets for all domains.";  
    input {  
      uses device-identify-infor;  
    }  
    output {  
      uses clock-current-DS-entry;  
    }  
  }  
  
  rpc read-clock-parent-DS {  
    description  
      "Reads the information about the PTP clock Parent  
      Datasets for all domains.";  
    input {  
      uses device-identify-infor;  
    }  
    output {  
      uses clock-parent-DS-entry;  
    }  
  }  
  
  rpc read-clock-default-DS {  
    description  
      "Reads the information about the PTP clock Parent  
      Datasets for all domains.";  
    input {  
      uses device-identify-infor;  
    }  
    output {  
      uses clock-default-DS-entry;  
    }  
  }  
  
  rpc read-clock-running{  
    description  
      "Reads the clock Running PTP information for a device.";  
    input {  
      uses device-identify-infor;  
    }  
    output {  
      uses clock-running-entry;  
    }  
  }
```

```
    }

    rpc read-clock-time-properties-DS{
      description
        "Reads the information about the PTP clock
        TimeProperties Datasets for all domains.";
      input {
        uses device-identify-infor;
      }
      output {
        uses clock-time-properties-DS-entry;
      }
    }

    rpc read-clock-trans-default-DS {
      description
        "Reads the information about the PTP clock Parent
        Datasets for all domains.";
      input {
        uses device-identify-infor;
      }
      output {
        uses clock-trans-default-DS-entry;
      }
    }
  }

  grouping port-identify-info{
    description "";

    leaf clock-identity {
      description "identify a specific PTP node.";
      config true;
      type binary {
        length "1..255";
      }
    }

    leaf port-number{
      description
        "This object specifies the PTP Portnumber for this
        port.";
      type uint32{
        range "0..65535";
      }
    }

    leaf domain-number{
```

```
        description
            "This object specifies the domain number used to create
            logical group of PTP devices.";
        type uint32{
            range "0..255";
        }
    }
}

grouping clock-port-entry{
    description
        "This group bundles together basic information about a
        single clock port.";

    leaf clock-port-name{
        description
            "This object specifies the PTP clock port name
            configured on the router.";
        type string;
    }

    leaf port-twostep-flag{
        description
            "This object specifies that two-step clock operation
            between the PTP master and slave device is enabled.";
        type boolean;
    }

    leaf clock-port-current-peer-address{
        description
            "This object specifies the current peer's network
            address used for PTP communication.";
        type string;
    }

    leaf clock-port-num-of-associated-ports{
        description
            "This object specifies - For a master port - the number
            of PTP slave sessions (peers) associated with this PTP
            port. For a slave port - the number of masters available
            to this slave port (might or might not be peered).";
        type uint32;
    }
}

grouping clock-port-DS-entry{
    description
```

"This group bundles together all information about the clock ports dataset for a single clock port.";

```
container port-identity{
  description
    "This object specifies the PTP clock port Identity,
    composed of clock-identity and portNumber.";
  leaf clock-identity {
    description "identify a specific PTP node.";
    config true;
    type binary {
      length "1..255";
    }
  }
  leaf port-number{
    description
      "This object specifies the PTP Portnumber for this
      port.";
    type uint32{
      range "0..65535";
    }
  }
}

leaf log-announcement-interval{
  description
    "This object specifies the Announce message transmission
    interval associated with this clock port.";
  type int64;
}

leaf announce-rct-timeout{
  description
    "This object specifies the Announce receipt timeout
    associated with this clock port.";
  type uint8;
}

leaf log-sync-interval{
  description
    "This object specifies the Sync message transmission
    interval.";
  type int8;
}

leaf min-delay-req-interval{
```

```
    description
      "This object specifies the Delay_Req message
      transmission interval.";
    type uint8;
  }

  leaf peer-delay-req-interval{
    description
      "This object specifies the Pdelay_Req message
      transmission interval.";
    type uint32;
  }

  leaf delay-mech{
    description
      "specifies the delay mechanism used. If the clock is an
      end-to-end clock, the value of the is e2e, else if the
      clock is a peer to-peer clock, the value shall be p2p.";
    type enumeration{
      enum E2E{
        value 01;
        description
          "The port is configured to use the delay request-
          response mechanism.";
      }
      enum P2P{
        value 02;
        description
          "The port is configured to use the peer delay
          mechanism.";
      }
      enum DISABLED{
        value 254;
        description
          "The port does not implement the delay mechanism.";
      }
    }
  }

  leaf peer-mean-path-delay{
    description
      "This object specifies the peer meanPathDelay.";
    type binary{
      length "1..256";
    }
  }
}
```

```
leaf grant-duration{
  description
    "This object specifies the grant duration allocated by
    the master.";
  type uint32;
}

leaf PTP-version{
  description
    "This object specifies the PTP version being used.";
  type uint32;
}
}

grouping clock-port-running-entry{
  description
    "This group bundles together all running dataset
    information about a single clock port.";

  leaf port-type{
    description
      "when the clock is configured in TCandBC mode, its port
      can be configured to TC or BC.";
    config true;
    when "device-clock-type='tcandbc' ";
    type enumeration{
      enum boundaryClock {
        value 2;
        description "configured as BC";
      }
      enum transparentClock {
        value 3;
        description "configured as TC";
      }
    }
  }
}

leaf running-state{
  description
    "This object specifies the port state returned by PTP
    engine. ";
  type enumeration{
    enum initializing{
      value 1;
      description
        "In this state a port initializes its data sets,
        hardware, and communication facilities.";
    }
  }
}
```

```
    }
    enum faulty{
        value 2;
        description "The fault state of the protocol.";
    }
    enum disabled{
        value 3;
        description
            "The port shall not place any messages on its
            communication path.";
    }
    enum listening{
        value 4;
        description
            "The port is waiting for the
            announceReceiptTimeout to expire or to receive an
            Announce message from a master.";
    }
    enum preMaster{
        value 5;
        description
            "The port shall behave in all respects as though
            it were in the MASTER state except that it shall
            not place any messages on its communication path
            except for Pdelay_Req, Pdelay_Resp,
            Pdelay_Resp_Follow_Up, signaling, or management
            messages.";
    }
    enum master{
        value 6;
        description "The port is behaving as a master port.";
    }
    enum passive{
        value 7;
        description
            "The port shall not place any messages on its
            communication path except for Pdelay_Req,
            Pdelay_Resp, Pdelay_Resp_Follow_Up, or signaling
            messages, or management messages that are a
            required response to another management message";
    }
    enum uncalibrated{
        value 8;
        description
            "The local port is preparing to synchronize to the
            master port.";
    }
}
```

```
        enum slave{
            value 9;
            description
                "The port is synchronizing to the selected master
                port.";
        }
    }
}
leaf running-role{
    description "This object specifies the Clock Role.";
    type enumeration{
        enum master{
            value 1;
            description
                "A clock that is the source of time to which all
                other clocks on that path synchronize.";
        }
        enum slave{
            value 2;
            description
                "A clock which synchronizes to another clock
                (master).";
        }
    }
}
leaf running-interface-index{
    description
        "This object specifies the interface on the router being
        used by the PTP Clock for PTP communication.";
    type uint32;
}

leaf running-Tx-mode{
    description "specifies the clock transmission mode ";
    type enumeration{
        enum unicast{
            value 1;
            description "Using unicast communication channel.";
        }
        enum multicast{
            value 2;
            description "Using Multicast communication channel.";
        }
        enum multicastmix{
            value 3;
            description
                "Using multicast-unicast communication channel.";
        }
    }
}
```



```
    }
  }
}

leaf running-Rx-mode{
  description "This object specifie the clock receive mode ";
  type enumeration{
    enum unicast{
      value 1;
      description "Using unicast communication channel.";
    }
    enum multicast{
      value 2;
      description "Using Multicast communication channel.";
    }
    enum multicastmix{
      value 3;
      description
        "Using multicast-unicast communication channel.";
    }
  }
}

leaf running-packets-received{
  description
    "This object specifies the packets received on the clock
    port (cummulative).";
  type binary{
    length "1..255";
  }
}

leaf running-packets-sent{
  description
    "This object specifies the packets sent on the clock
    port (cummulative).";
  type uint64;
}

grouping clock-port-trans-DS-entry{
  description
    "This group bundles together all clock port Transparent
    dataset information about a single clock port.";

  leaf log-min-pdelay-req-int{
    description
      "This object specifies the value of the logarithm to the
      base 2 of the minPdelayReqInterval.";
  }
}
```

```
    type int32{
      range "-128..127";
    }
  }
  leaf faulty-flag{
    description
      "This object specifies the value TRUE if the port is
      faulty and FALSE if the port is operating normally.";
    type boolean;
  }
  leaf peer-mean-path-delay{
    description
      "This object specifies, (if the delayMechanism used is
      P2P) the value is the estimate of the current one-way
      propagation delay, i.e., <meanPathDelay> on the link
      attached to this port, computed using the peer delay
      mechanism. If the value of the delayMechanism used is
      E2E, then the value will be zero.";
    type binary{
      length "1..255";
    }
  }
}

grouping clock-port-associate-entry{
  description
    "This group bundles together all information about a
    single associated port for the given clockport.";

  leaf port-index{
    description
      "This object specifies the associated port's serial
      number in the current port's context.";
    type uint32{
      range "1..65535";
    }
  }

  leaf associate-address{
    description
      "This object specifies the peer port's network address
      used for PTP communication.";
    type binary{
      length "1..255";
    }
  }
  leaf associate-packets-sent{
```

```
    description
      "The number of packets sent to this peer port from the
      current port.";
    type binary{
      length "1..255";
    }
  }
  leaf associate-packets-received{
    description
      "The number of packets received from this peer port by
      the current port.";
    type binary{
      length "1..255";
    }
  }
  leaf associate-in-errors{
    description
      "This object specifies the input errors associated with
      the peer port.";
    type binary{
      length "1..255";
    }
  }
  leaf associate-out-errors{
    description
      "This object specifies the output errors associated with
      the peer port.";
    type binary{
      length "1..255";
    }
  }
}

container port-ptp-query-table{
  list ptp-port-query{
    description "";

    key "clock-identity port-number domain-number";
    uses port-identify-info;
    container clock-port{
      uses clock-port-entry;
    }
    container clock-port-DS{
      uses clock-port-DS-entry;
    }
    container clock-port-running{
      uses clock-port-running-entry;
    }
  }
}
```

```
    }
    container clock-port-trans-DS{
        uses clock-port-trans-DS-entry;
    }
    container clock-port-associate{
        uses clock-port-associate-entry;
    }
}
}

rpc read-port-ptp-query-table{
    description
        "Read all port-level ptp information of a specific port
        on a specific device in a specific domain.";
    input{
        uses port-identify-info;
    }
    output{
        container clock-port{
            uses clock-port-entry;
        }
        container clock-port-DS{
            uses clock-port-DS-entry;
        }
        container clock-port-running{
            uses clock-port-running-entry;
        }
        container clock-port-trans-DS{
            uses clock-port-trans-DS-entry;
        }
        container clock-port-associate{
            uses clock-port-associate-entry;
        }
    }
}

rpc read-clock-port {
    description
        "Reads the information about the PTP clock Parent
        Datasets for all domains.";
    input {
        uses port-identify-info;
    }
    output {
        uses clock-port-entry;
    }
}
```

```
rpc read-clock-port-DS{
  description
    "Reads the information about the PTP clock Parent
    Datasets for all domains.";
  input {
    uses port-identify-info;
  }
  output {
    uses clock-port-DS-entry;
  }
}

rpc read-clock-port-running{
  description
    "Reads the information about the PTP clock Parent
    Datasets for all domains.";
  input {
    uses port-identify-info;
  }
  output {
    uses clock-port-running-entry;
  }
}

rpc read-clock-port-trans-DS{
  description
    "Reads the information about the PTP clock Parent
    Datasets for all domains.";
  input {
    uses port-identify-info;
  }
  output {
    uses clock-port-trans-DS-entry;
  }
}

rpc read-clock-port-associate{
  description
    "Reads the information about the PTP clock Parent
    Datasets for all domains.";
  input {
    uses port-identify-info;
  }
  output {
    uses clock-port-associate-entry;
  }
}
}
```

6. Security Considerations

YANG modules are designed to be accessed via the NETCONF protocol [RFC6241], thus security considerations in [RFC6241] apply here. Security measures such as using the NETCONF over SSH [RFC6242] and restricting its use with access control [RFC6536] can further improve its security, avoid injection attacks and misuse of the protocol.

Some data nodes defined in this YANG module are writable, and any changes to them may adversely impact a synchronization network.

7. IANA Considerations

This document registers a URI in the IETF XML registry, and the following registration is requested to be made:

URI: urn:ietf:params:xml:ns:yang:1588v2

This document registers a YANG module in the YANG Module Names:

name: 1588v2 namespace: urn:ietf:params:xml:ns:yang:1588v2

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997
- [RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF) ", RFC 6020, October 2010
- [RFC6991] Schoenwaelder, J., "Common YANG Data Types", RFC 6991, July 2013
- [IEEE1588] IEEE, "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems", IEEE Std 1588-2008, July 2008

8.2. Informative References

- [RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011

- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, March 2012
- [PTP-MIB] Shankarkumar, V., Montini, L., Frost, T., and Dowd, G., "Precision Time Protocol Version 2 (PTPv2) Management Information Base", draft-ietf-tictoc-ntp-mib-07, Work in progress

9. Acknowledgments

TBD

Authors' Addresses

Yuanlong Jiang
Huawei Technologies Co., Ltd.
Bantian, Longgang district
Shenzhen 518129, China
Email: jiangyuanlong@huawei.com

Xian Liu
Huawei Technologies Co., Ltd.
Bantian, Longgang district
Shenzhen 518129, China

Jinchun Xu
Huawei Technologies Co., Ltd.
Bantian, Longgang district
Shenzhen 518129, China

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 29, 2015

N. Wu
A. Kumar S N
Huawei
June 27, 2015

A YANG Data Model for NTP
draft-wu-ntp-ntp-cfg-01

Abstract

This document defines a YANG data model for Network Time Protocol implementations. The data model includes configuration data and state data.

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].

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 <http://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 December 29, 2015.

Copyright Notice

Copyright (c) 2015 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 (<http://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. Terminology	2
1.2. Tree Diagrams	3
2. NTP data model	3
3. Relationship to NTPv4-MIB	5
4. NTP YANG Module	7
5. IANA Considerations	22
6. Security Considerations	22
7. Acknowledgments	23
8. References	23
8.1. Normative References	23
8.2. Informative References	23
Authors' Addresses	23

1. Introduction

This document defines a YANG [RFC6020] data model for Network Time Protocol [RFC5905] implementations.

The data model covers configuration of system parameters of NTP, such as access rules, authentication and VRF binding, and also associations of NTP in different modes and parameters of per-interface. It also provides information about running state of NTP implementations.

1.1. Terminology

The following terms are defined in [RFC6020]:

- o configuration data
- o data model
- o module
- o state data

The terminology for describing YANG data models is found in [RFC6020].

1.2. Tree Diagrams

A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is as follows:

- o Brackets "[" and "]" enclose list keys.
- o Abbreviations before data node names: "rw" means configuration data (read-write), and "ro" means state data (read-only).
- o Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.
- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- o Ellipsis ("...") stands for contents of subtrees that are not shown.

2. NTP data model

This document defines the YANG module "ietf-ntp", which has the following structure:

```

module: ietf-ntp
  +--rw ntp-cfg!
  |   +--rw ntp-enabled?          boolean
  |   +--rw refclock-master
  |   |   +--rw master?          boolean
  |   |   +--rw master-stratum?  ntp-stratum
  |   +--rw authentication!
  |   |   +--rw auth-enabled?     boolean
  |   |   +--rw trusted-key?      uint32
  |   |   +--rw authentication-keys* [key-id]
  |   |   |   +--rw key-id        uint32
  |   |   |   +--rw algorithm?    enumeration
  |   |   |   +--rw password?     union
  |   +--rw access-rules
  |   |   +--rw access-rule* [access-mode]
  |   |   |   +--rw access-mode    enumeration
  |   |   |   +--rw acl-number
  |   |   |   |   +--rw (acl-type)?
  |   |   |   |   |   +--:(ipv4)
  |   |   |   |   |   |   +--rw acl-number-ipv4?  uint16
  |   |   |   |   |   +--:(ipv6)
  |   |   |   |   |   |   +--rw acl-number-ipv6?  uint16
  |   +--rw associations

```

```

+--rw peers
|   +--rw peer* [address vrf]
|       +--rw version?    ntp-version
|       +--rw address     inet:ip-address
|       +--rw key-id?    leafref
|       +--rw minpoll?   ntp-minpoll
|       +--rw maxpoll?   ntp-maxpoll
|       +--rw prefer?    boolean
|       +--rw burst?     boolean
|       +--rw iburst?    boolean
|       +--rw vrf        string
|       +--rw source?    leafref
+--rw servers
|   +--rw server* [address vrf]
|       +--rw version?    ntp-version
|       +--rw address     inet:ip-address
|       +--rw key-id?    leafref
|       +--rw minpoll?   ntp-minpoll
|       +--rw maxpoll?   ntp-maxpoll
|       +--rw prefer?    boolean
|       +--rw burst?     boolean
|       +--rw iburst?    boolean
|       +--rw vrf        string
|       +--rw source?    leafref
+--rw ntp-interfaces
|   +--rw ntp-interface* [ntp-ifname]
|       +--rw ntp-ifname    leafref
|       +--rw multicast-client
|           |   +--rw multicast-client-address?    union
|           +--rw multicast-server
|               |   +--rw multicast-server-address?    inet:ip-address
|               |   +--rw multicast-server-ttl?        uint8
|               |   +--rw multicast-server-version?    ntp-version
|               |   +--rw multicast-server-keyid?      leafref
|           +--rw broadcast-client
|               |   +--rw broadcast-client-enabled?    boolean
|           +--rw broadcast-server
|               +--rw broadcast-server-version?    ntp-version
|               +--rw broadcast-server-keyid?      leafref
+--ro ntp-state
|   +--ro system-status
|       +--ro clock-state?    enumeration
|       +--ro clock-stratum?  ntp-stratum
|       +--ro clock-refid?    union
|       +--ro nominal-freq?    decimal64
|       +--ro actual-freq?     decimal64
|       +--ro clock-precision? uint8
|       +--ro clock-offset?    decimal64

```

```

|   |--ro root-delay?           decimal64
|   |--ro root-dispersion?     decimal64
|   |--ro peer-dispersion?     decimal64
|   |--ro reference-time?      string
|   |--ro sync-state?          enumeration
|--ro associations-status
|   |--ro association-status* [association-source]
|   |--ro association-source    union
|   |--ro association-stratum?  ntp-stratum
|   |--ro association-refid?    union
|   |--ro association-reach?    uint8
|   |--ro association-poll?     uint8
|   |--ro association-now?      uint32
|   |--ro association-offset?   decimal64
|   |--ro association-delay?    decimal64
|   |--ro association-dispersion? decimal64
|   |--ro association-sent?     uint32
|   |--ro association-sent-fail? uint32
|   |--ro association-received? uint32
|   |--ro association-dropped?  uint32
|--ro ntp-statistics
|   |--ro packet-sent?          uint32
|   |--ro packet-sent-fail?    uint32
|   |--ro packet-received?     uint32
|   |--ro packet-dropped?      uint32

```

This data model defines two primary containers, one for NTP configuration and the other is for NTP running state. The NTP configuration container includes data nodes for access rules, authentication, associations and interfaces. In the NTP running state container, there are data nodes for system status and associations.

3. Relationship to NTPv4-MIB

If the device implements the NTPv4-MIB [RFC5907], data nodes in container ntp-cfg and ntp-state from YANG module can be mapped to table entries in NTPv4-MIB.

The following tables list the YANG data nodes with corresponding objects in the NTPv4-MIB.

YANG data nodes in /ntp-cfg/	NTPv4-MIB objects
ntp-enabled	ntpEntStatusCurrentMode

YANG data nodes in /ntp-cfg/associations/peers/peer /ntp-cfg/associations/servers/server	NTPv4-MIB objects
address	ntpAssocAddressType ntpAssocAddress

YANG NTP Configuration Data Nodes and Related NTPv4-MIB Objects

YANG data nodes in /ntp-state/system-status	NTPv4-MIB objects
clock-state clock-stratum clock-refid	ntpEntStatusCurrentMode ntpEntStatusStratum ntpEntStatusActiveRefSourceId
clock-precision clock-offset root-dispersion	ntpEntStatusActiveRefSourceName ntpEntTimePrecision ntpEntStatusActiveOffset ntpEntStatusDispersion

YANG data nodes in /ntp-state/associations-status/ association-status/	NTPv4-MIB objects
association-source	ntpAssocAddressType ntpAssocAddress
association-stratum	ntpAssocStratum
association-refid	ntpAssocRefId
association-offset	ntpAssocOffset
association-delay	ntpAssocStatusDelay
association-dispersion	ntpAssocStatusDispersion
association-sent	ntpAssocStatOutPkts
association-received	ntpAssocStatInPkts
association-dropped	ntpAssocStatProtocolError

YANG NTP State Data Nodes and Related NTPv4-MIB Objects

4. NTP YANG Module

```
//<CODE BEGINS> file "ietf-ntp@2015-06-27.yang"

module ietf-ntp {

  namespace "urn:ietf:params:xml:ns:yang:ietf-ntp";

  prefix "ntp";

  import ietf-inet-types {
    prefix "inet";
  }

  import ietf-interfaces {
    prefix "if";
  }

  organization
    "IETF NTP (Network Time Protocol) Working Group";

  contact
    "WG Web: <http://tools.ietf.org/wg/ntp/>
    WG List: <mailto:ntpwg@lists.ntp.org>
    WG Chair: Karen O'Donoghue
              <mailto:odonoghue@isoc.org>
    Editor:   Eric Wu
              <mailto:eric.wu@huawei.com>";
    Editor:   Anil Kumar S N
              <mailto:anil.sn@huawei.com>";

  description
    "This YANG module defines essential components for the
    management of a routing subsystem.

    Copyright (c) 2014 IETF Trust and the persons identified
    as authors of the code. All rights reserved.

    Redistribution and use in source and binary forms,
    with or without modification, is permitted pursuant to,
    and subject to the license terms contained in, the
    Simplified BSD License set forth in Section 4.c of the
    IETF Trust's Legal Provisions Relating to IETF Documents
    (http://trustee.ietf.org/license-info).

    This version of this YANG module is part of RFC XXXX;
    see the RFC itself for full legal notices.";

  revision 2015-06-27 {
    description
```

```
    "Updated revision.";
  reference
    "RFC XXXX: A YANG Data Model for NTP Management";
}

/* Typedef Definitions */
typedef ntp-stratum {
  type uint8;
  description
    "The level of each server in the hierarchy is defined by a
    stratum number. Primary servers are assigned stratum one;
    secondary servers at each lower level are assigned stratum
    numbers one greater than the preceding level";
}

typedef ntp-version {
  type uint8 {
    range "1..4";
  }
  default "3";
  description
    "The current NTP version supported by corresponding
    association.";
}

typedef ntp-minpoll {
  type uint8 {
    range "4..17";
  }
  default "6";
  description
    "The minimul poll interval for this NTP association.";
}

typedef ntp-maxpoll {
  type uint8 {
    range "4..17";
  }
  default "10";
  description
    "The maximul poll interval for this NTP association.";
}

typedef multicast-client-v4address {
```



```
    type inet:ipv4-address;
    default "224.0.1.1";
    description
      "The IPv4 address for NTP multicast client.";
  }

typedef multicast-client-v6address {
  type inet:ipv6-address;
  default "FF0E::0101";
  description
    "The IPv6 address for NTP multicast client.";
}

/* Groupings */
grouping authentication-key {
  description
    "To define an authentication key for a Network Time Protocol
    (NTP) time source.";
  leaf key-id {
    type uint32 {
      range "1..max";
    }
    description
      "Authentication key identifier.";
  }
  leaf algorithm {
    type enumeration {
      enum md5 {
        description
          "Message Digest 5 (MD5) algorithm.";
      }
      enum hmac-sha256 {
        description
          "Secure Hash Algorithm 256 algorithm.";
      }
    }
    description
      "Authentication algorithm.";
  }
  leaf password {
    type union {
      type string {
        length "1..255";
      }
      type string {
        length "20..392";
      }
    }
  }
}
```

```
    }
    description "Clear or encrypted mode for password text.";
  }
}

grouping association-param {
  description
    "To define parameters for a Network Time Protocol (NTP)
    associations.";
  leaf version {
    type ntp-version;
    description
      "NTP version.";
  }
  leaf address {
    type inet:ip-address;
    description
      "The IP address of this association.";
  }
  leaf key-id {
    type leafref {
      path "/ntp:ntp-cfg/ntp:authentication/"
        + "ntp:authentication-keys/ntp:key-id";
    }
    description
      "Authentication key id referenced in this association.";
  }
  leaf minpoll {
    type ntp-minpoll;
    description
      "The minimul poll interval used in this association.";
  }
  leaf maxpoll {
    type ntp-maxpoll;
    description
      "The maximul poll interval used in this association.";
  }
  leaf prefer {
    type boolean;
    default "false";
    description
      "Whether this association is preferred.";
  }
  leaf burst {
    type boolean;
    default "false";
    description
      "Sends a series of packets instead of a single packet";
  }
}
```

```
        within each synchronization interval to achieve faster
        synchronization.";
    }
    leaf iburst {
        type boolean;
        default "false";
        description
            "Sends a series of packets instead of a single packet
            within the initial synchronization interval to achieve
            faster initial synchronization.";
    }
    leaf vrf {
        type string;
        description
            "The VRF instance this association binded to.";
    }
    leaf source {
        type leafref {
            path "/if:interfaces/if:interface/if:name";
        }
        description
            "The interface whose ip address this association used
            as source address.";
    }
}
}
```

```
/* Configuration data nodes */
container ntp-cfg {
    presence
        "Enables NTP unless the 'ntp-enabled' leaf
        (which defaults to 'true') is set to 'false'";
    description
        "Configuration parameters for NTP.";
    leaf ntp-enabled {
        type boolean;
        default true;
        description
            "Controls whether NTP is enabled or disabled
            on this device.";
    }
}
```

```
container refclock-master {
    description
        "Configuration for reference clock.";
    leaf master {
        type boolean;
        default false;
    }
}
```

```
        description
            "Use its own NTP master clock to synchronize with peers
            when true.";
    }
    leaf master-stratum {
        type ntp-stratum;
        default "16";
        description
            "Use its own NTP master clock to synchronize with peers
            when true.";
    }
}

container authentication {
    presence
        "Enables NTP authentication when the 'auth-enabled'
        leaf is set to 'true'.";
    description
        "Configuration of authentication.";
    leaf auth-enabled {
        type boolean;
        default false;
        description
            "Controls whether NTP authentication is enabled
            or disabled on this device.";
    }
    leaf trusted-key {
        type uint32;
        description
            "The key trusted by NTP.";
    }
    list authentication-keys {
        key "key-id";
        uses authentication-key;
        description
            "List of authentication key.";
    }
}

container access-rules {
    description
        "Configuration of access rules.";
    list access-rule {
        key "access-mode";
        description
            "List of access rules.";
        leaf access-mode {
            type enumeration {
```

```
enum peer {
  description
    "Sets the fully access authority. Both time
    request and control query can be performed
    on the local NTP service, and the local clock
    can be synchronized to the remote server.";
}
enum server {
  description
    "Enables the server access and query.
    Both time requests and control query can be
    performed on the local NTP service, but the
    local clock cannot be synchronized to the
    remote server.";
}
enum synchronization {
  description
    "Enables the server to access.
    Only time request can be performed on the
    local NTP service.";
}
enum query {
  description
    "Sets the maximum access limitation.
    Control query can be performed only on the
    local NTP service.";
}
}
description
  "NTP access mode.";
}
container acl-number {
  description
    "Configuration of acl numbers.";
  choice acl-type {
    description
      "Type of acl.";
    case ipv4 {
      leaf acl-number-ipv4 {
        type uint16;
        description "IPv4 acl number.";
      }
    }
    case ipv6 {
      leaf acl-number-ipv6 {
        type uint16;
        description "IPv6 acl number.";
      }
    }
  }
}
```

```
    }
  }
}

container associations {
  description
    "Configuration of association.";
  container peers {
    description
      "Peer associations.";
    list peer {
      key "address vrf";
      uses association-param;
      description
        "List of peers.";
    }
  }
  container servers {
    description
      "Sever associations.";
    list server {
      key "address vrf";
      uses association-param;
      description
        "List of servers.";
    }
  }
}

container ntp-interfaces {
  description
    "Configuration parameters for NTP interfaces.";
  list ntp-interface {
    key "ntp-ifname";
    description
      "List of interfaces.";
    leaf ntp-ifname {
      type leafref {
        path "/if:interfaces/if:interface/if:name";
      }
      description
        "The interface name.";
    }
  }
  container multicast-client {
    description
      "Configuration of multicast-client.";
  }
}
```

```
    leaf multicast-client-address {
      type union {
        type multicast-client-v4address;
        type multicast-client-v6address;
      }
      description
        "The IP address of the multicast group to join.";
    }
  }
  container multicast-server {
    description
      "Configuration of multicast-server.";
    leaf multicast-server-address {
      type inet:ip-address;
      description
        "The IP address to send NTP multicast packets.";
    }
    leaf multicast-server-ttl {
      type uint8;
      description
        "Specifies the time to live (TTL) of a multicast
        packet.";
    }
    leaf multicast-server-version {
      type ntp-version;
      description
        "Specifies the version a multicast packet.";
    }
    leaf multicast-server-keyid {
      type leafref {
        path "/ntp:ntp-cfg/ntp:authentication/"
          + "ntp:authentication-keys/ntp:key-id";
      }
      description
        "Specifies the authentication key id of a
        multicast packet.";
    }
  }
}
container broadcast-client {
  description
    "Configuration of broadcast-client.";
  leaf broadcast-client-enabled {
    type boolean;
    description
      "Allows a device to receive Network Time Protocol
      (NTP) broadcast packets on an interface.";
  }
}
```

```

    container broadcast-server {
      description
        "Configuration of broadcast-server.";
      leaf broadcast-server-version {
        type ntp-version;
        description
          "Specifies the version of a broadcast packet.";
      }
      leaf broadcast-server-keyid {
        type leafref {
          path "/ntp:ntp-cfg/ntp:authentication/"
            + "ntp:authentication-keys/ntp:key-id";
        }
        description
          "Specifies the authentication key id of a
          broadcast packet.";
      }
    }
  }
}
}
}
}

```

```

/* Operational state data */

```

```

container ntp-state {
  config "false";
  description
    "Operational state of the NTP.";

  container system-status {
    description
      "System status of NTP.";
    leaf clock-state {
      type enumeration {
        enum synchronized {
          description
            "Indicates that the local clock has been
            synchronized with an NTP server or
            the reference clock.";
        }
        enum unsynchronized {
          description
            "Indicates that the local clock has not been
            synchronized with any NTP server.";
        }
      }
    }
    description "Indicates the state of system clock.";
  }
  leaf clock-stratum {

```



```
    type ntp-stratum;
    description
      "Indicates the stratum of the reference clock.";
  }
  leaf clock-refid {
    type union {
      type inet:ipv4-address;
      type binary {
        length "4";
      }
      type string {
        length "4";
      }
    }
    description
      "IPv4 address or first 32 bits of the MD5 hash of
      the IPv6 address or reference clock of the peer to
      which clock is synchronized.";
  }
  leaf nominal-freq {
    type decimal64 {
      fraction-digits 4;
    }
    description
      "Indicates the nominal frequency of the
      local clock, in Hz.";
  }
  leaf actual-freq {
    type decimal64 {
      fraction-digits 4;
    }
    description
      "Indicates the actual frequency of the
      local clock, in Hz.";
  }
  leaf clock-precision {
    type uint8;
    description
      "Precision of the clock of this system
      in Hz.(prec=2(-n))";
  }
  leaf clock-offset {
    type decimal64 {
      fraction-digits 4;
    }
    description
      "Offset of clock to synchronized peer,
      in milliseconds.";
```

```
    }
    leaf root-delay {
      type decimal64 {
        fraction-digits 2;
      }
      description
        "Total delay along path to root clock,
        in milliseconds.";
    }
    leaf root-dispersion {
      type decimal64 {
        fraction-digits 2;
      }
      description
        "Indicates the dispersion between the local clock
        and the master reference clock, in milliseconds.";
    }
    leaf peer-dispersion {
      type decimal64 {
        fraction-digits 2;
      }
      description
        "Indicates the dispersion between the local clock
        and the peer clock, in milliseconds.";
    }
    leaf reference-time {
      type string;
      description
        "Indicates reference timestamp.";
    }
    leaf sync-state {
      type enumeration {
        enum clock-not-set {
          description
            "Indicates the clock is not updated.";
        }
        enum freq-set-by-cfg {
          description
            "Indicates the clock frequency is set by
            NTP configuration.";
        }
        enum clock-set {
          description
            "Indicates the clock is set.";
        }
        enum freq-not-determined {
          description
            "Indicates the clock is set but the frequency
```

```
        is not determined.";
    }
    enum clock-synchronized {
        description
            "Indicates that the clock is synchronized.";
    }
    enum spike {
        description
            "Indicates a time difference of more than 128
            milliseconds is detected between NTP server
            and client clock. The clock change will take
            effect in XXX seconds.";
    }
}
description
    "Indicates the synchronization status of
    the local clock.";
}
}

container associations-status {
    description
        "System status of NTP.";
    list association-status {
        key "association-source";
        description
            "List of association status.";
        leaf association-source {
            type union {
                type inet:ipv4-address;
                type inet:ipv6-address;
            }
            description
                "IPv4 or IPv6 address of the peer. If a
                nondefault VRF is configured for the peer,
                the VRF follows the address.";
        }
        leaf association-stratum {
            type ntp-stratum;
            description
                "Indicates the stratum of the reference clock.";
        }
        leaf association-refid {
            type union {
                type inet:ipv4-address;
                type binary {
                    length "4";
                }
            }
        }
    }
}
```

```
        type string {
            length "4";
        }
    }
    description
        "Reference clock type or address for the peer.";
}
leaf association-reach {
    type uint8;
    description
        "Indicates the reachability of the configured
        server or peer.";
}
leaf association-poll {
    type uint8;
    description
        "Indicates the polling interval for current,
        in seconds.";
}
leaf association-now {
    type uint32;
    description
        "Indicates the time since the NTP packet was
        not received or last synchronized, in seconds.";
}
leaf association-offset {
    type decimal64 {
        fraction-digits 4;
    }
    description
        "Indicates the offset between the local clock
        and the superior reference clock.";
}
leaf association-delay {
    type decimal64 {
        fraction-digits 2;
    }
    description
        "Indicates the delay between the local clock
        and the superior reference clock.";
}
leaf association-dispersion {
    type decimal64 {
        fraction-digits 2;
    }
    description
        "Indicates the dispersion between the local
        clock and the superior reference clock.";
```

```
    }
    leaf association-sent {
      type uint32;
      description
        "Indicates the total number of packets this
        association sent.";
    }
    leaf association-sent-fail {
      type uint32;
      description
        "Indicates the number of times packet sending
        failed by this association.";
    }
    leaf association-received {
      type uint32;
      description
        "Indicates the total number of packets
        this association received.";
    }
    leaf association-dropped {
      type uint32;
      description
        "Indicates the number of packets
        this association dropped.";
    }
  }
}

container ntp-statistics {
  description
    "Packet statistics of NTP.";
  leaf packet-sent {
    type uint32;
    description
      "Indicates the total number of packets sent.";
  }
  leaf packet-sent-fail {
    type uint32;
    description
      "Indicates the number of times packet
      sending failed.";
  }
  leaf packet-received {
    type uint32;
    description
      "Indicates the total number of packets received.";
  }
  leaf packet-dropped {
```

```
        type uint32;
        description
            "Indicates the number of packets dropped.";
    }
}
}
}
//<CODE ENDS>
```

5. IANA Considerations

This document registers a URI in the "IETF XML Registry" [RFC3688]. Following the format in RFC 3688, the following registration has been made.

URI: urn:ietf:params:xml:ns:yang:ietf-ntp

Registrant Contact: The NETMOD WG of the IETF.

XML: N/A; the requested URI is an XML namespace.

This document registers a YANG module in the "YANG Module Names" registry [RFC6020].

Name: ietf-ntp

Namespace: urn:ietf:params:xml:ns:yang:ietf-ntp

Prefix: ntp

Reference: RFC XXXX

6. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [RFC6242]. The NETCONF access control model [RFC6536] provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

There are a number of data nodes defined in the YANG module which are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., <edit-config>) to these data nodes without proper protection can have a negative effect on network operations.

7. Acknowledgments

TBD.

8. References

8.1. Normative References

- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.
- [RFC5905] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.
- [RFC5907] Gerstung, H., Elliott, C., and B. Haberman, "Definitions of Managed Objects for Network Time Protocol Version 4 (NTPv4)", RFC 5907, June 2010.
- [RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.
- [RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, March 2012.

8.2. Informative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

Authors' Addresses

Nan Wu
Huawei
Huawei Bld., No.156 Beiqing Rd.
Beijing 100095
China

Email: eric.wu@huawei.com

Anil Kumar S N
Huawei
Kundalahalli Village, Whitefield
Bangalore, Kanataka 560037
India

Email: anil.sn@huawei.com