Network Working Group
Internet-Draft
Intended status: Informational
Expires: August 25, 2021

N. Rozen-Schiff
D. Dolev
Hebrew University of Jerusalem
T. Mizrahi
Huawei Network.IO Innovation Lab
M. Schapira
Hebrew University of Jerusalem
February 21, 2021

A Secure Selection and Filtering Mechanism for the Network Time Protocol
draft-ietf-ntp-chronos-02

Abstract

The Network Time Protocol version 4 (NTPv4), as defined in RFC 5905, is the mechanism used by NTP clients to synchronize with NTP servers across the Internet. This document specifies an extension to the NTPv4 client, named Chronos, which is used as a "watchdog" alongside NTPv4, and provides improved security against time shifting attacks. Chronos involves changes to the NTP client's system process only and is backwards compatible with NTPv4 servers. Chronos is also compatible with NTPv5, since it does not affect the wire protocol.

Status of This Memo

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

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

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

This Internet-Draft will expire on August 25, 2021.

Copyright Notice

Table of Contents

1. Introduction

NTPv4, as defined in RFC 5905 [RFC5905], is vulnerable to time shifting attacks, in which the attacker's goal is to shift the local time at an NTP client. See [Chronos_paper] for details. Time shifting attacks on NTP are possible even if NTP communication is encrypted and authenticated. A weaker man-in-the-middle (MitM) attacker can shift time simply by dropping or delaying packets, whereas a powerful attacker, who has full control over an NTP server, can determine the response content. This document introduces a time shifting mitigation mechanism called Chronos. Chronos is backwards compatible with NTPv4 and serves as an NTPv4 client's "watchdog" for time shifting attacks. An NTP client that runs Chronos is interoperable with [RFC5905]-compatible NTPv4 servers. Chronos is also compatible with NTPv5, since it does not affect the wire protocol.

Chronos is a background mechanism that continuously maintains a virtual "Chronos" clock update and compares it to NTPv4's clock

update. When the gap between the two updates exceeds a certain threshold (specified in Section 6), this is interpreted as the client experiencing a time shifting attack. In this case, Chronos is used to update the client's clock, and NTPv4 is operated in the background until the gap between NTPv4 and Chronos' updates are again below this threshold, and hence NTPv4 is safe to use again.

Due to Chronos operating in the background, the client clock's precision and accuracy are precisely as in NTPv4 while not experiencing a time-shifting attack. When under attack, Chronos prevents the clock from being shifted by the attacker, thus still preserving high accuracy and precision (as discussed in Section 6).

Chronos achieves accurate synchronization even in the presence of powerful attackers who are in direct control of a large number of NTP servers: up to 1/3 of the servers in the pool (where the pool may consist of hundreds or even thousands of servers). NTPv4 chooses a small subset of the NTP server pool (e.g. 4 servers), and periodically queries this subset of servers. Thus, even if only 1/3 of the servers in the pool are compromised, the small subset that is used by NTPv4 may consist of a majority of faulty servers. Conversely, Chronos constantly updates the set of servers it queries; in each poll interval Chronos randomly chooses a different subset of servers from the pool. Thus, even if an attack is not detected in a given poll interval, Chronos is bound to detect the attack within a relatively small number of poll intervals.

A Chronos client iteratively "crowdsources" time queries across NTP servers and applies a provably secure algorithm for eliminating "suspicious" responses and for averaging over the remaining responses. Chronos is carefully engineered to minimize communication overhead so as to avoid overloading NTP servers. Chronos' security was evaluated both theoretically and experimentally with a prototype implementation. These evaluation results indicate that in order to successfully shift time at a Chronos client by over 100 milliseconds from the UTC, even a powerful man-in-the-middle attacker requires over 20 years of effort in expectation. The full paper is available at [Chronos_paper].

Chronos introduces a watchdog mechanism that is added to the client's system process and maintains a virtual clock value that is used as a reference for detecting attacks. The virtual clock value computation differs from the current NTPv4 in two key aspects. First, a Chronos client relies on a large number of NTP servers, from which only few servers to synchronize with are periodically chosen at random, in order to avoid overloading the servers. Second, the selection algorithm of the virtual clock uses an approximate agreement technique to remove outliers, thus limiting the attacker's ability to

contaminate the "time samples" (offsets) derived from the queried NTP servers. These two elements of Chronos' design provide provable security guarantees against both man-in-the-middle attackers and attackers capable of compromising a large number of NTP servers.

2. Conventions Used in This Document

2.1. 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 RFC 2119 [RFC2119].

2.2. Terms and Abbreviations

NTPv4	Network Time Protocol version 4 [RFC5905].
Selection process	Clock filter algorithm and system process [RFC5905].

2.3. Notations

Describing Chronos algorithm, the following notation are used.

Notation	Meaning
n	The number of candidate servers in the pool that Chronos can query (potentially hundreds)
m	The number of servers that NTPv4 queries in each poll interval (up to tens)
w	An upper bound on the distance of the local time from the UTC at any NTP server with an accurate clock (termed "truechimer" in [RFC5905])
Cest	The client's estimation for the time that has passed since its last synchronization to the server pool (sec)
B	An upper bound on the client's time estimation error (ms/sec)
ERR	An upper bound on the client's error regarding his estimation of the time passed from the last update, equals to B*Cest (ms)
K	Panic trigger - the number of pool re-sampling until reaches "Panic mode";
tc	The current time [sec], as indicated by the virtual clock value that is computed by Chronos

Table 1: Chronos Notations

The recommended values are discussed in Section 3.2.

3. Extension to the NTP System Process

A client that runs Chronos as a watchdog, uses NTPv4 as in [RFC5905] and in the background runs a modification to the elements of the system process described in Section 11.2.1 and 11.2.2 in [RFC5905] (namely, the Selection Algorithm and the Cluster Algorithm). The NTPv4 conventional protocol periodically queries m servers in each poll interval. In parallel the Chronos watchdog periodically queries a (variable) set of m servers in each Chronos poll interval. Specifically, in Chronos, after executing the "Clock Filter Algorithm" as defined in Section 10 in [RFC5905], the client discards outliers by executing the procedure described in this section and the next. Then, the NTPv4 "Combine Algorithm" is used for computing the system peer offset, as specified in Section 11.2.3 in [RFC5905]. In each poll interval the Chronos virtual clock value is compared with the NTPv4 clock value, and if the difference exceeds a predetermined value, an attack is detected. This process holds also for Chronos as a watchdog of future NTPv5.

3.1. Chronos' System Process

At the first time the Chronos system process is executed, calibration is needed. The calibration process generates a local pool of servers the client can synchronize with, consisting of n servers (up to hundreds). To this end, the NTP client executes the "Peer Process" and "Clock Filter Algorithm" as in Sections 9,10 in [RFC5905] (respectively), on an hourly basis, for 24 consecutive hours, and generates the union of all received NTP servers' IP addresses. Importantly, this process can also be executed in the background periodically, once in a long time (e.g., every few weeks/months).

In each Chronos poll interval the Chronos system process randomly chooses a set of m servers (where n with magnitude of hundreds and m of tens) out of the local pool of n servers. Then, out of the time-samples received from this chosen subset of servers, a lowest third of the samples' offset values and highest third of the samples' offset values are discarded.

Chronos checks that the following two conditions hold for the remaining samples:

- The maximal distance between every two time samples does not exceed 2w.

- The average value of the remaining samples is at distance at most ERR+2w from the client's local clock (as computed by Chronos).

(where w, ERR are as described in Table 1. Notice that ERR magnitude is approximately LAMBDA as defined in [RFC5905]).

In the event that both of these conditions are satisfied, the average of the remaining samples is the "final offset". Otherwise, a random partial of the interval is chosen, after which a new subset of servers is sampled, in the exact same manner. This way, Chronos client queries are spread across the time interval better in case of DoS attack on the NTP servers. This resampling process continues in subsequent Chronos poll intervals until the two conditions are both satisfied or the number of times the servers are re-sampled exceeds a "Panic Trigger" (K in Table 1), in which case, Chronos enters a "Panic Mode". Note that it is configurable whether the client allows panic mode or not.

In panic mode, Chronos queries all the servers in the local server pool, orders the collected time samples from lowest to highest and eliminates the bottom third and the top third of the samples. The client then averages over the remaining samples, and sets this average to be the new "final offset".

As in [RFC5905], the final offset is passed on to the clock discipline algorithm for the purpose of steering the Chronos virtual clock to the correct time. The Chronos virtual clock is then compared to the NTPv4 (or to the future NTPv5) clock as part of the watchdog process.

3.2. Chronos' Recommended Parameters

According to empirical observations (presented in [Chronos_paper]), querying 15 servers at each poll interval (i.e., m=15) out of 500 servers (n=500), and setting w to be around 25 milliseconds provides both high time accuracy and good security. Moreover, empirical analyses showed that, on average, approximately 83% of the servers' clocks are at most w-away from the UTC, and within 2w from each other, satisfying the first condition of Chronos' system process.

Furthermore, according to Chronos security analysis, setting K to be 3 (i.e., if after 3 re-sampling, the two conditions are not satisfied, then Chronos reaches "panic mode") is both safe when facing time shifting attacks and the probability of reaching the "panic mode" is negligible (less than 0.000002).

Chronos effect on precision and accuracy are discussed in Section 5 and Section 6.

4. Chronos' Pseudocode

The pseudocode for Chronos' Time Sampling Scheme, which is invoked in each Chronos poll interval is as follows:

```
counter := 0
S = []
T = []
While counter < K do
   S := sample(m) //gather samples from (tens of) randomly chosen servers
   T := bi-side-trim(S,1/3) //trim the third lowest and highest values
   if (max(T) -min(T) <= 2w) and (|avg(T)-tc| < ERR + 2w) Then
       return avg(t)
   end
   counter ++
   sleep(rand(0,1)*poll interval)
end
// panic mode
S := sample(n)
T := bi-sided-trim(S,1/3) //trim bottom and top thirds;
return avg(T)
```

5. Precision vs. Security

Since NTPv4 (and future NTPv5) updates the clock as long as time-shifting attacks are not detected, the precision and accuracy of a Chronos client are the same as NTPv4 when not under attack. Under attack, Chronos, which changes the list of the sampled servers more frequently than NTPv4 [Chronos_paper], and does not use some of the filters in NTPv4's system process, can potentially be less precise (though provably more secure than NTPv4, which is vulnerable to time-shifting attacks [RFC5905]).

However, our experimental and empirical analyses of Chronos revealed that Chornos and NTPv4 exhibit the same level of precision and accuracy when not under attack, with Chronos maintaining this level even in the presence of time-shifting attacks.

6. Chronos' Threat Model and Security Guarantees

As explained above, Chronos repeatedly gathers time samples from small subsets of a large local pool of NTP servers. The following form of a man-in-the-middle (MitM) Byzantine attacker is considered: the MitM attacker is assumed to control a subset of the servers in the local pool of servers and is capable of determining precisely the values of the time samples gathered by the Chronos client from these NTP servers. The threat model thus encompasses a broad spectrum of MitM attackers, ranging from fairly weak (yet dangerous) MitM attackers only capable of delaying and dropping packets to extremely powerful MitM attackers who are in control of (even authenticated) NTP servers. MitM attackers captured by this framework might be, for example, (1) in direct control of a fraction of the NTP servers (e.g., by exploiting a software vulnerability), (2) an ISP (or other Autonomous-System-level attacker) on the default BGP paths from the NTP client to a fraction of the available servers, (3) a nation state with authority over the owners of NTP servers in its jurisdiction, or (4) an attacker capable of hijacking (e.g., through DNS cache poisoning or BGP prefix hijacking) traffic to some of the available NTP servers. The details of the specific attack scenario are abstracted by reasoning about MitM attackers in terms of the fraction of servers with respect to which the attacker has MitM capabilities.

Chronos detects time-shifting attacks by constantly monitoring NTPv4's (or NTPv5's) offset and the offset computed by Chronos, as explained above, and checking whether it exceeds a certain threshold (10 milliseconds by default).

Analytical results (in [Chronos_paper]) indicate that in order to succeed in shifting time at a Chronos client by even a small amount (e.g., 100 milliseconds), even a powerful MitM attacker requires many

years of effort (e.g., over 20 years in expectation). See a brief overview of Chronos' security analysis below.

Notably, Chronos provides protection from MitM attacks that cannot be achieved by cryptographic authentication protocols since even with such measures in place an attacker can still influence time by dropping/delaying packets. However, adding an authentication and crypto-based security layer to Chronos will enhance its security guarantees and enable the detection of various spoofing and modification attacks.

Chronos' security analysis is briefly described next.

6.1. Security Analysis Overview

Time-samples that are at most w away from the UTC are considered "good", whereas other samples are considered "malicious". Two scenarios are considered:

- Less than 2/3 of the queried servers are under the attacker's control.
- The attacker controls more than 2/3 of the queried servers.

The first scenario, where there are more than 1/3 good samples, consists of two sub-cases: (i) there is at least one good sample in the set of samples not eliminated by Chronos (that is, in the middle third of samples), and (ii) there are no good samples in the remaining set of samples. In the first of these two cases (at least one good sample in the set of samples was not eliminated by Chronos), the other remaining samples, including those provided by the attacker, must be close to a good sample (for otherwise, the first condition of Chronos' system process in Section 3.1 is violated and a new set of servers is chosen). This implies that the average of the remaining samples must be close to the UTC. In the second case (there are no good samples in the set of remaining samples), since more than a third of the initial samples were good, both the (discarded) third lowest-value samples and the (discarded) third highest-value samples must each contain a good sample. Hence, all the remaining samples are bounded from both above and below by good samples, and so is their average value, implying that this value is close to the UTC [RFC5905].

In the second scenario, where the attacker controls more than 2/3 of the queried servers, the worst possibility for the client is that all remaining samples are malicious (i.e., more than w away from the UTC). However, as proved in [Chronos_paper], the probability of this scenario is extremely low even if the attacker controls a large

fraction (e.g., 1/4) of the servers in the local pool. The probability that the attacker repeatedly succeeds in realising this scenario decays exponentially, rendering the probability of a significant time shift negligible. See [Chronos_paper] for details.

Beyond evaluating the probability of an attacker successfully shifting time at the client's clock, we also evaluated the probability that the attacker succeeds in launching a DoS attack on the servers by causing many clients to enter panic mode (and so query all the servers in their local pools). This probability too is negligible even for an attacker in control of a large number of servers in clients' local server pools. See [Chronos_paper]for details.

Further details about Chronos's threat model and security guarantees can be found in [Chronos_paper].

7. Acknowledgements

The authors would like to thank Erik Kline, Miroslav Lichvar, Danny Mayer, Karen O'Donoghue, Dieter Sibold, Yaakov. J. Stein, and Harlan Stenn, for valuable contributions to this document and helpful discussions and comments.

8. IANA Considerations

This memo includes no request to IANA.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

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

9.2. Informative References

[Chronos_paper]
Deutsch, O., Schiff, N., Dolev, D., and M. Schapira, "Preventing (Network) Time Travel with Chronos", 2018, <https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_02A-2_Deutsch_paper.pdf>.

Authors' Addresses

Neta Rozen-Schiff
Hebrew University of Jerusalem
Jerusalem
Israel

Phone: +972 2 549 4599
Email: neta.r.schiff@gmail.com

Danny Dolev
Hebrew University of Jerusalem
Jerusalem
Israel

Phone: +972 2 549 4588
Email: danny.dolev@mail.huji.ac.il

Tal Mizrahi
Huawei Network.IO Innovation Lab
Israel

Email: tal.mizrahi.phd@gmail.com

Michael Schapira
Hebrew University of Jerusalem
Jerusalem
Israel

Phone: +972 2 549 4570
Email: schapiram@huji.ac.il

Internet Engineering Task Force
Internet-Draft
Updates: 5905 (if approved)
Intended status: Standards Track
Expires: March 21, 2021

M. Lichvar
Red Hat
A. Malhotra
Boston University

Sep 17, 2020

NTP Interleaved Modes
draft-ietf-ntp-interleaved-modes-04

Abstract

This document extends the specification of Network Time Protocol (NTP) version 4 in RFC 5905 with special modes called the NTP interleaved modes, that enable NTP servers to provide their clients and peers with more accurate transmit timestamps that are available only after transmitting NTP packets. More specifically, this document describes three modes: interleaved client/server, interleaved symmetric, and interleaved broadcast.

Status of This Memo

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

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

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

This Internet-Draft will expire on March 21, 2021.

Copyright Notice

Table of Contents

1. Introduction

RFC 5905 [RFC5905] describes the operations of NTPv4 in a client/server, symmetric, and broadcast mode. The transmit and receive timestamps are two of the four timestamps included in every NTPv4 packet used for time synchronization.

For a highly accurate and stable synchronization, the transmit and receive timestamp should be captured close to the beginning of the actual transmission and the end of the reception respectively. An asymmetry in the timestamping causes the offset measured by NTP to have an error.

There are at least four options where a timestamp of an NTP packet may be captured with a software NTP implementation running on an operating system:

1. User space (software)

2. Network device driver or kernel (software)

3. Data link layer (hardware - MAC chip)

4. Physical layer (hardware - PHY chip)

Software timestamps captured in the user space in the NTP implementation itself are least accurate. They do not include system calls used for sending and receiving packets, processing and queuing

delays in the system, network device drivers, and hardware. Hardware timestamps captured at the physical layer are most accurate.

A transmit timestamp captured in the driver or hardware is more accurate than the user-space timestamp, but it is available to the NTP implementation only after it sent the packet using a system call. The timestamp cannot be included in the packet itself unless the driver or hardware supports NTP and can modify the packet before or during the actual transmission.

The protocol described in RFC 5905 does not specify any mechanism for a server to provide its clients and peers with a more accurate transmit timestamp that is known only after the transmission. A packet that strictly follows RFC 5905, i.e. it contains a transmit timestamp corresponding to the packet itself, is said to be in basic mode.

Different mechanisms could be used to exchange timestamps known after the transmission. The server could respond to each request with two packets. The second packet would contain the transmit timestamp corresponding to the first packet. However, such a protocol would enable a traffic amplification, or it would use packets with an asymmetric length, which would cause an asymmetry in the network delay and an error in the measured offset.

This document describes an interleaved client/server, interleaved symmetric, and interleaved broadcast mode. In these modes, the server sends a single packet, which contains a transmit timestamp corresponding to the previous packet that was sent to the client or peer. This transmit timestamp can be captured at any of the four places mentioned above. Both servers and clients/peers are required to keep some state specific to the interleaved mode.

The protocol does not change the NTP packet header format. Only the semantics of some timestamp fields is different. NTPv4 that supports client/server and broadcast interleaved modes is compatible with NTPv4 without this capability as well as with all previous NTP versions.

This document assumes familiarity with RFC 5905.

1.1. Requirements Language

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

2. Interleaved Client/server mode

The interleaved client/server mode is similar to the basic client/server mode. The only difference between the two modes is in the meaning of the transmit and origin timestamp fields.

A client request in the basic mode has an origin timestamp equal to the transmit timestamp from the previous server response, or is zero. A server response in the basic mode has an origin timestamp equal to the transmit timestamp from the client's request. The transmit timestamps correspond to the packets in which they are included.

A client request in the interleaved mode has an origin timestamp equal to the receive timestamp from the previous server response. A server response in the interleaved mode has an origin timestamp equal to the receive timestamp from the client's request. The transmit timestamps correspond to the previous packets that were sent to the server or client.

A server which supports the interleaved mode needs to save pairs of local receive and transmit timestamps. The server SHOULD discard old timestamps to limit the amount of memory needed to support clients using the interleaved mode. The server MAY separate the timestamps by IP addresses, but it SHOULD NOT separate them by port numbers, i.e. clients are allowed to change their source port between requests.

The server MAY restrict the interleaved mode to specific IP addresses and/or authenticated clients.

Both servers and clients that support the interleaved mode MUST NOT send a packet that has a transmit timestamp equal to the receive timestamp in order to reliably detect whether received packets conform to the interleaved mode.

The transmit and receive timestamps in server responses need to be unique to prevent two different clients from sending requests with the same origin timestamp and the server responding in the interleaved mode with an incorrect transmit timestamp. If the timestamps are not guaranteed to be monotonically increasing, the server SHOULD check that the transmit and receive timestamp is not already saved as a receive timestamp of a previous request (from the same IP address if the server separates timestamps by addresses), and generate a new timestamp if necessary.

When the server receives a request from a client, it SHOULD respond in the interleaved mode if the following conditions are met:

1. The request does not have a receive timestamp equal to the transmit timestamp.

2. The origin timestamp from the request matches the local receive timestamp of a previous request that the server has saved (for the IP address if it separates timestamps by addresses).

A response in the interleaved mode MUST contain the transmit timestamp of the response which contained the receive timestamp matching the origin timestamp from the request. The server SHOULD drop the timestamps after sending the response. The receive timestamp MUST NOT be used again to detect a request conforming to the interleaved mode.

If the conditions are not met, the server MUST NOT respond in the interleaved mode. The server MAY always respond in the basic mode. In any case, the server SHOULD save the new receive and transmit timestamps.

The first request from a client is always in the basic mode and so is the server response. It has a zero origin timestamp and zero receive timestamp. Only when the client receives a valid response from the server, it will be able to send a request in the interleaved mode.

The protocol recovers from packet loss. When a client request or server response is lost, the client will use the same origin timestamp in the next request. The server can respond in the interleaved mode if it still has the timestamps corresponding to the origin timestamp. If the server already responded to the timestamp in the interleaved mode, or it had to drop the timestamps for other reasons, it will respond in the basic mode and save new timestamps, which will enable an interleaved response to the following request. The client SHOULD limit the number of requests in the interleaved mode between server responses to prevent processing of very old timestamps in case a large number of consecutive requests is lost.

An example of packets in a client/server exchange using the interleaved mode is shown in Figure 1. The packets in the basic and interleaved mode are indicated with B and I respectively. The timestamps t1', t3' and t11' point to the same transmissions as t1, t3 and t11, but they may be less accurate. The first exchange is in the basic mode followed by a second exchange in the interleaved mode. For the third exchange, the client request is in the interleaved mode, but the server response is in the basic mode, because the server did not have the pair of timestamps t6 and t7 (e.g. they were dropped to save timestamps for other clients using the interleaved mode).

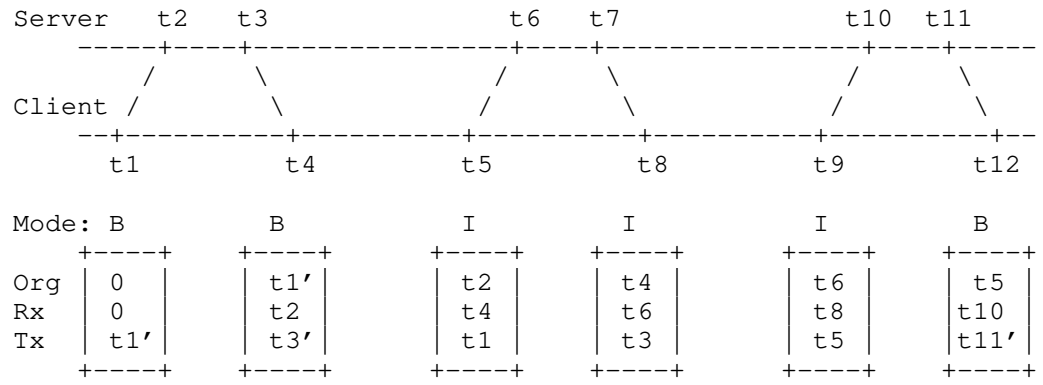

Figure 1: Packet timestamps in interleaved client/server mode

When the client receives a response from the server, it performs the tests described in RFC 5905. Two of the tests are modified for the interleaved mode:

1. The check for duplicate packets SHOULD compare both receive and transmit timestamps in order to not drop a valid response in the interleaved mode if it follows a response in the basic mode and they contain the same transmit timestamp.

2. The check for bogus packets SHOULD compare the origin timestamp with both transmit and receive timestamps from the request. If the origin timestamp is equal to the transmit timestamp, the response is in the basic mode. If the origin timestamp is equal to the receive timestamp, the response is in the interleaved mode.

The client SHOULD NOT update its NTP state when an invalid response is received to not lose the timestamps which will be needed to complete a measurement when the following response in the interleaved mode is received.

If the packet passed the tests and conforms to the interleaved mode, the client can compute the offset and delay using the formulas from RFC 5905 and one of two different sets of timestamps. The first set is RECOMMENDED for clients that filter measurements based on the delay. The corresponding timestamps from Figure 1 are written in parentheses.

T1 - local transmit timestamp of the previous request (t1)

T2 - remote receive timestamp from the previous response (t2)

T3 - remote transmit timestamp from the latest response (t3)

T4 - local receive timestamp of the previous response (t4)

The second set gives a more accurate measurement of the current offset, but the delay is much more sensitive to a frequency error between the server and client due to a much longer interval between T1 and T4.

T1 - local transmit timestamp of the latest request (t5)

T2 - remote receive timestamp from the latest response (t6)

T3 - remote transmit timestamp from the latest response (t3)

T4 - local receive timestamp of the previous response (t4)

Clients MAY filter measurements based on the mode. The maximum number of dropped measurements in the basic mode SHOULD be limited in case the server does not support or is not able to respond in the interleaved mode. Clients that filter measurements based on the delay will implicitly prefer measurements in the interleaved mode over the basic mode, because they have a shorter delay due to a more accurate transmit timestamp (T3).

The server MAY limit saving of the receive and transmit timestamps to requests which have an origin timestamp specific to the interleaved mode in order to not waste resources on clients using the basic mode. Such an optimization will delay the first interleaved response of the server to a client by one exchange.

A check for a non-zero origin timestamp works with clients that implement NTP data minimization [I-D.ietf-ntp-data-minimization]. To detect requests in the basic mode from clients that do not implement the data minimization, the server can encode in low-order bits of the receive and transmit timestamps below precision of the clock a bit indicating whether the timestamp is a receive timestamp. If the server receives a request with a non-zero origin timestamp which does not indicate it is a receive timestamp of the server, the request is in the basic mode and it is not necessary to save the new receive and transmit timestamp.

3. Interleaved Symmetric mode

The interleaved symmetric mode uses the same principles as the interleaved client/server mode. A packet in the interleaved symmetric mode has a transmit timestamp which corresponds to the

previous packet sent to the peer and an origin timestamp equal to the receive timestamp from the last packet received from the peer.

In order to prevent the peer from matching the transmit timestamp with an incorrect packet when the peers' transmissions do not alternate (e.g. they use different polling intervals) and a previous packet was lost, the use of the interleaved mode in symmetric associations requires additional restrictions.

Peers which have an association need to count valid packets received between their transmissions to determine in which mode a packet should be formed. A valid packet in this context is a packet which passed all NTP tests for duplicate, replayed, bogus, and unauthenticated packets. Other received packets may update the NTP state to allow the (re)initialization of the association, but they do not change the selection of the mode.

A peer A SHOULD send a peer B a packet in the interleaved mode only when the following conditions are met:

1. The peer A has an active association with the peer B which was specified with an option enabling the interleaved mode, OR the peer A received at least one valid packet in the interleaved mode from the peer B.

2. The peer A did not send a packet to the peer B since it received the last valid packet from the peer B.

3. The previous packet that the peer A sent to the peer B was the only response to a packet received from the peer B.

An example of packets exchanged in a symmetric association is shown in Figure 2. The minimum polling interval of the peer A is twice as long as the maximum polling interval of the peer B. The first packets sent by the peers are in the basic mode. The second and third packet sent by the peer A is in the interleaved mode. The second packet sent by the peer B is in the interleaved mode, but the following packets sent by the peer are in the basic mode, because multiple responses are sent per request.

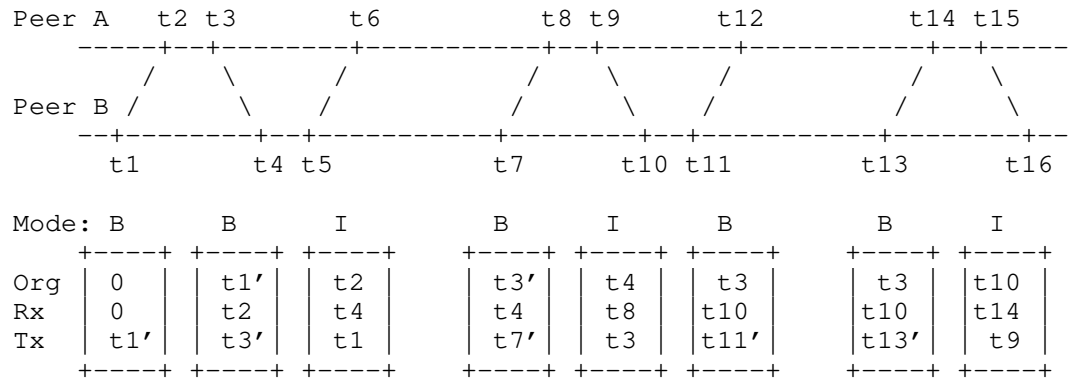

Figure 2: Packet timestamps in interleaved symmetric mode

If the peer A has no association with the peer B and it responds with symmetric passive packets, it does not need to count the packets in order to meet the restrictions, because each request has at most one response. The peer SHOULD process the requests in the same way as a server which supports the interleaved client/server mode. It MUST NOT respond in the interleaved mode if the request was not in the interleaved mode.

The peers SHOULD compute the offset and delay using one the two sets of timestamps specified in the client/server section. They MAY switch between them to minimize the interval between T1 and T4 in order to reduce the error in the measured delay.

4. Interleaved Broadcast mode

A packet in the interleaved broadcast mode contains two transmit timestamps. One corresponds to the packet itself and is saved in the transmit timestamp field. The other corresponds to the previous packet and is saved in the origin timestamp field. The packet is compatible with the basic mode, which uses a zero origin timestamp.

An example of packets sent in the broadcast mode is shown in Figure 3.

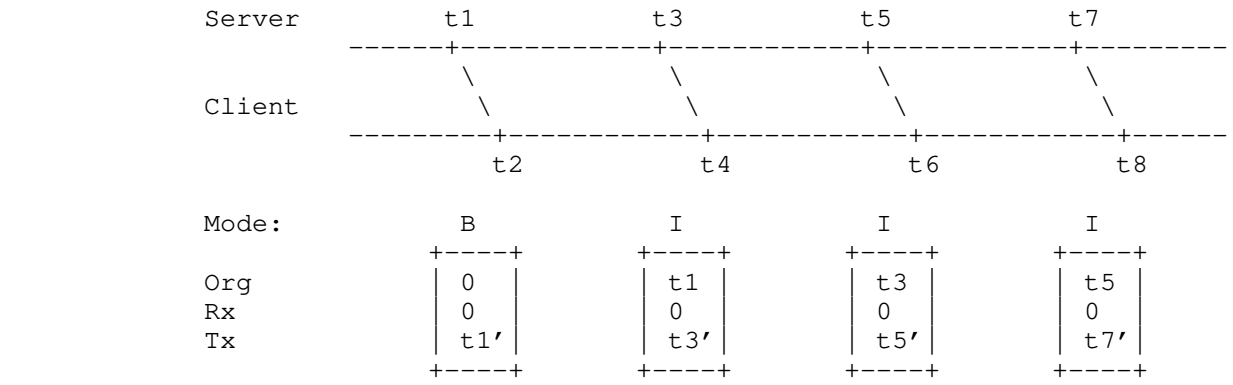

Figure 3: Packet timestamps in interleaved broadcast mode

A client which does not support the interleaved mode ignores the origin timestamp and processes all packets as if they were in the basic mode.

A client which supports the interleaved mode SHOULD check if the origin timestamp is not zero to detect packets in the interleaved mode. The client SHOULD also compare the origin timestamp with the transmit timestamp from the previous packet to detect lost packets. If the difference is larger than a specified maximum (e.g. 1 second), the packet SHOULD NOT be used for synchronization.

The client SHOULD compute the offset using the origin timestamp from the received packet and the local receive timestamp of the previous packet. If the client needs to measure the network delay, it SHOULD use the interleaved client/server mode.

5. Acknowledgements

The interleaved modes described in this document are based on the implementation written by David Mills in the NTP project [1]. The specification of the broadcast mode is based purely on this implementation. The specification of the symmetric mode has some modifications. The client/server mode is specified as a new mode compatible with the symmetric mode, similarly to the basic symmetric and client/server modes.

The authors would like to thank Daniel Franke, Tal Mizrahi, Steven Sommars, Harlan Stenn, and Kristof Teichel for their useful comments.

6. IANA Considerations

This memo includes no request to IANA.

7. Security Considerations

The security considerations of time protocols in general are discussed in RFC 7384 [RFC7384], and specifically the security considerations of NTP are discussed in RFC 5905.

Security issues that apply to the basic modes apply also to the interleaved modes. They are described in The Security of NTP's Datagram Protocol [SECNTP].

Clients and peers SHOULD NOT leak the receive timestamp in packets sent to other peers or clients (e.g. as a reference timestamp) to prevent off-path attackers from easily getting the origin timestamp needed to make a valid response in the interleaved mode.

Clients using the interleaved mode SHOULD randomize all bits of both receive and transmit timestamps, as recommended for the transmit timestamp in the NTP client data minimization [I-D.ietf-ntp-data-minimization], to make it more difficult for off-path attackers to guess the origin timestamp. It is not possible to zero the origin timestamp to prevent passive observers from easily tracking clients moving between different networks.

Attackers can force the server to drop its timestamps in order to prevent clients from getting an interleaved response. They can send a large number of requests, send requests with a spoofed source address, or replay an authenticated request if the interleaved mode is enabled only for authenticated clients. Clients SHOULD NOT rely on servers to be able to respond in the interleaved mode.

Protecting symmetric associations in the interleaved mode against replay attacks is even more difficult than in the basic mode. The NTP state needs to be protected not only between the reception and transmission in order to send the peer a packet with a valid origin timestamp, but all the time to not lose the timestamps which will be needed to complete a measurement when the following packet in the interleaved mode is received.

8. References

8.1. Normative References

[I-D.ietf-ntp-data-minimization]
Franke, D. and A. Malhotra, "NTP Client Data Minimization", draft-ietf-ntp-data-minimization-04 (work in progress), March 2019.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

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

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

8.2. Informative References

[RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, DOI 10.17487/RFC7384, October 2014, <https://www.rfc-editor.org/info/rfc7384>.

[SECNTP] Malhotra, A., Gundy, M., Varia, M., Kennedy, H., Gardner, J., and S. Goldberg, "The Security of NTP's Datagram Protocol", 2016, <http://eprint.iacr.org/2016/1006>.

8.3. URIs

[1] http://www.ntp.org

Authors' Addresses

Miroslav Lichvar
Red Hat
Purkynova 115
Brno 612 00
Czech Republic

Email: mlichvar@redhat.com

Aanchal Malhotra
Boston University
111 Cummington St
Boston 02215
USA

Email: aanchal4@bu.edu

Network Time Protocol (ntp) Working Group
Internet-Draft
Updates: 5905 (if approved)
Intended status: Standards Track
Expires: March 19, 2021

F. Gont
G. Gont
SI6 Networks
M. Lichvar
Red Hat

September 15, 2020

Port Randomization in the Network Time Protocol Version 4
draft-ietf-ntp-port-randomization-06

Abstract

The Network Time Protocol can operate in several modes. Some of these modes are based on the receipt of unsolicited packets, and therefore require the use of a well-known port as the local port number. However, in the case of NTP modes where the use of a well-known port is not required, employing such well-known port unnecessarily increases the ability of attackers to perform blind/off-path attacks. This document formally updates RFC5905, recommending the use of transport-protocol ephemeral port randomization for those modes where use of the NTP well-known port is not required.

Table of Contents

1. Introduction

The Network Time Protocol (NTP) is one of the oldest Internet protocols, and currently specified in [RFC5905]. Since its original implementation, standardization, and deployment, a number of vulnerabilities have been found both in the NTP specification and in some of its implementations [NTP-VULN]. Some of these vulnerabilities allow for off-path/blind attacks, where an attacker can send forged packets to one or both NTP peers for achieving Denial of Service (DoS), time-shifts, or other undesirable outcomes. Many of these attacks require the attacker to guess or know at least a target NTP association, typically identified by the tuple {srcaddr, srcport, dstaddr, dstport, keyid} (see section 9.1 of [RFC5905]). Some of these parameters may be easily known or guessed.

NTP can operate in several modes. Some of these modes rely on the ability of nodes to receive unsolicited packets, and therefore require the use of the NTP well-known port (123). However, for modes where the use of a well-known port is not required, employing the NTP well-known port improves the ability of an attacker to perform blind/

off-path attacks (since knowledge of the port numbers is typically required for such attacks). A recent study [NIST-NTP] that analyzes the port numbers employed by NTP clients suggests that a considerable number of NTP clients employ the NTP well-known port as their local port, or select predictable ephemeral port numbers, thus improving the ability of attackers to perform blind/off-path attacks against NTP.

BCP 156 [RFC6056] already recommends the randomization of transport-protocol ephemeral ports. This document aligns NTP with the recommendation in BCP 156 [RFC6056], by formally updating [RFC5905] such that port randomization is employed for those NTP modes for which the use of the NTP well-known port is not needed.

2. Terminology

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

3. Considerations About Port Randomization in NTP

The following subsections analyze a number of considerations about transport-protocol ephemeral port randomization when applied to NTP.

3.1. Mitigation Against Off-path Attacks

There has been a fair share of work in the area of off-path/blind attacks against transport protocols and upper-layer protocols, such as [RFC5927] and [RFC4953]. Whether the target of the attack is a transport protocol instance (e.g., TCP connection) or an upper-layer protocol instance (e.g., an application protocol instance), the attacker is required to know or guess the five-tuple {Protocol, IP Source Address, IP Destination Address, Source Port, Destination Port} that identifies the target transport protocol instance or the transport protocol instance employed by the target upper-layer protocol instance. Therefore, increasing the difficulty of guessing this five-tuple helps mitigate blind/off-path attacks.

As a result of these considerations, BCP 156 [RFC6056] recommends the randomization of transport-protocol ephemeral ports. Thus, this document aims to bring the NTP specification [RFC5905] in line with the aforementioned recommendation.

We note that the use of port randomization is a transport-layer mitigation against off-path/blind attacks, and does not preclude (nor

is it precluded by) other possible mitigations for off-path attacks that might be implemented by an application protocol (e.g. [I-D.ietf-ntp-data-minimization]). For instance, some of the aforementioned mitigations may be ineffective against some off-path attacks [NTP-FRAG] or may benefit from the additional entropy provided by port randomization [NTP-security].

3.2. Effects on Path Selection

Intermediate systems implementing the Equal-Cost Multi-Path (ECMP) algorithm may select the outgoing link by computing a hash over a number of values, that include the transport-protocol source port. Thus, as discussed in [NTP-CHLNG], the selected client port may have an influence on the measured offset and delay.

If the source port is changed with each request, packets in different exchanges will be more likely to take different paths, which could cause the measurements to be less stable and have a negative impact on the stability of the clock.

Network paths to/from a given server are less likely to change between requests if port randomization is applied on a per-association basis. This approach minimizes the impact on the stability of NTP measurements, but may cause different clients in the same network synchronized to the same NTP server to have a significant stable offset between their clocks due to their NTP exchanges consistently taking different paths with different asymmetry in the network delay.

Section 4 recommends NTP implementations to randomize the ephemeral port number of client/server associations. The choice of whether to randomize the port number on a per-association or a per-request basis is left to the implementation.

3.3. Filtering of NTP traffic

In a number of scenarios (such as when mitigating DDoS attacks), a network operator may want to differentiate between NTP requests sent by clients, and NTP responses sent by NTP servers. If an implementation employs the NTP well-known port for the client port number, requests/responses cannot be readily differentiated by inspecting the source and destination port numbers. Implementation of port randomization for non-symmetrical modes allows for simple differentiation of NTP requests and responses, and for the enforcement of security policies that may be valuable for the mitigation of DDoS attacks, when all NTP clients in a given network employ port randomization.

3.4. Effect on NAT devices

Some NAT devices will not translate the source port of a packet when a privileged port number is employed. In networks where such NAT devices are employed, use of the NTP well-known port for the client port will essentially limit the number of hosts that may successfully employ NTP client implementations.

In the case of NAT devices that will translate the source port even when a privileged port is employed, packets reaching the external realm of the NAT will not employ the NTP well-known port as the local port, since the local port will normally be translated by the NAT device possibly, but not necessarily, with a random port.

3.5. Relation to Other Mitigations for Off-Path Attacks

Transport-protocol ephemeral port randomization is a best current practice (BCP 156) that helps mitigate off-path attacks at the transport-layer. It is orthogonal to other possible mitigations for off-path attacks that may be implemented at other layers (such as the use of timestamps in NTP) which may or may not be effective against some off-path attacks (see e.g. [NTP-FRAG]. This document aligns NTP with the existing best current practice on ephemeral port selection, irrespective of other techniques that may (and should) be implemented for mitigating off-path attacks.

4. Update to RFC5905

The following text from Section 9.1 ("Peer Process Variables") of [RFC5905]:

> dstport: UDP port number of the client, ordinarily the NTP port number PORT (123) assigned by the IANA. This becomes the source port number in packets sent from this association.

is replaced with:

> dstport: UDP port number of the client. In the case of broadcast server mode (5) and symmetric modes (1 and 2), it SHOULD contain the NTP port number PORT (123) assigned by the IANA. In the client mode (3), it SHOULD contain a randomized port number, as specified in [RFC6056]. The value in this variable becomes the source port number of packets sent from this association. The randomized port number SHOULD NOT be shared with other associations.
>
> If a client implementation performs ephemeral port randomization on a per-request basis, it SHOULD close the corresponding socket/

port after each request/response exchange. In order to prevent duplicate or delayed server packets from eliciting ICMP port unreachable error messages at the client, the client MAY wait for more responses from the server for a specific period of time (e.g. 3 seconds) before closing the UDP socket/port.

NOTES:

The choice of whether to randomize the ephemeral port number on a per-request or a per-association basis is left to the implementation, and should consider the possible effects on path selection along with its possible impact on time measurement.

On most current operating systems, which implement ephemeral port randomization [RFC6056], an NTP client may normally rely on the operating system to perform ephemeral port randomization. For example, NTP implementations using POSIX sockets may achieve ephemeral port randomization by *not* binding the socket with the bind() function, or binding it to port 0, which has a special meaning of "any port". connect()ing the socket will make the port inaccessible by other systems (that is, only packets from the specified remote socket will be received by the application).

5. Implementation Status

[RFC Editor: Please remove this section before publication of this document as an RFC.]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

OpenNTPD:

[OpenNTPD] has never explicitly set the local port of NTP clients, and thus employs the ephemeral port selection algorithm implemented by the operating system. Thus, on all operating

systems that implement port randomization (such as current versions of OpenBSD, Linux, and FreeBSD), OpenNTPD will employ port randomization for client ports.

chrony:
[chrony] by default does not set the local client port, and thus employs the ephemeral port selection algorithm implemented by the operating system. Thus, on all operating systems that implement port randomization (such as current versions of OpenBSD, Linux, and FreeBSD), chrony will employ port randomization for client ports.

nwtime.org's sntp client:
sntp does not explicitly set the local port, and thus employs the ephemeral port selection algorithm implemented by the operating system. Thus, on all operating systems that implement port randomization (such as current versions of OpenBSD, Linux, and FreeBSD), it will employ port randomization for client ports.

6. IANA Considerations

There are no IANA registries within this document. The RFC-Editor can remove this section before publication of this document as an RFC.

7. Security Considerations

The security implications of predictable numeric identifiers [I-D.irtf-pearg-numeric-ids-generation] (and of predictable transport-protocol port numbers [RFC6056] in particular) have been known for a long time now. However, the NTP specification has traditionally followed a pattern of employing common settings and code even when not strictly necessary, which at times has resulted in negative security and privacy implications (see e.g. [I-D.ietf-ntp-data-minimization]). The use of the NTP well-known port (123) for the srcport and dstport variables is not required for all operating modes. Such unnecessary usage comes at the expense of reducing the amount of work required for an attacker to successfully perform off-path/blind attacks against NTP. Therefore, this document formally updates [RFC5905], recommending the use of transport-protocol port randomization when use of the NTP well-known port is not required.

This issue has been tracked by US-CERT with VU#597821, and has been assigned CVE-2019-11331.

8. Acknowledgments

The authors would like to thank (in alphabetical order) Ivan Arce, Dhruv Dhody, Todd Glassey, Watson Ladd, Aanchal Malhotra, Danny Mayer, Gary E. Miller, Tomoyuki Sahara, Dieter Sibold, Steven Sommars, Kristof Teichel, and Ulrich Windl, for providing valuable comments on earlier versions of this document.

Watson Ladd raised the problem of DDoS mitigation when the NTP well-known port is employed as the client port (discussed in Section 3.3 of this document).

The authors would like to thank Harlan Stenn for answering questions about nwtime.org's NTP implementation.

Fernando would like to thank Nelida Garcia and Jorge Oscar Gont, for their love and support.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

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

[RFC6056] Larsen, M. and F. Gont, "Recommendations for Transport-Protocol Port Randomization", BCP 156, RFC 6056, DOI 10.17487/RFC6056, January 2011, <https://www.rfc-editor.org/info/rfc6056>.

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

9.2. Informative References

[chrony] "chrony", <https://chrony.tuxfamily.org/>.

[I-D.ietf-ntp-data-minimization]
Franke, D. and A. Malhotra, "NTP Client Data Minimization", draft-ietf-ntp-data-minimization-04 (work in progress), March 2019.

[I-D.irtf-pearg-numeric-ids-generation]
Gont, F. and I. Arce, "On the Generation of Transient Numeric Identifiers", draft-irtf-pearg-numeric-ids-generation-03 (work in progress), September 2020.

[NIST-NTP]
Sherman, J. and J. Levine, "Usage Analysis of the NIST Internet Time Service", Journal of Research of the National Institute of Standards and Technology Volume 121, March 2016, <https://tf.nist.gov/general/pdf/2818.pdf>.

[NTP-CHLNG]
Sommars, S., "Challenges in Time Transfer Using the Network Time Protocol (NTP)", Proceedings of the 48th Annual Precise Time and Time Interval Systems and Applications Meeting, Monterey, California pp. 271-290, January 2017, <http://leapsecond.com/ntp/NTP_Paper_Sommars_PTTI2017.pdf>.

[NTP-FRAG]
Malhotra, A., Cohen, I., Brakke, E., and S. Goldberg, "Attacking the Network Time Protocol", NDSS'17, San Diego, CA. Feb 2017, 2017, <http://www.cs.bu.edu/~goldbe/papers/NTPattack.pdf>.

[NTP-security]
Malhotra, A., Van Gundy, M., Varia, V., Kennedy, H., Gardner, J., and S. Goldberg, "The Security of NTP's Datagram Protocol", Cryptology ePrint Archive Report 2016/1006, 2016, <https://eprint.iacr.org/2016/1006>.

[NTP-VULN]
Network Time Foundation, "Security Notice", Network Time Foundation's NTP Support Wiki , <https://support.ntp.org/bin/view/Main/SecurityNotice>.

[OpenNTPD]
"OpenNTPD Project", <https://www.openntpd.org>.

[RFC4953] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, DOI 10.17487/RFC4953, July 2007, <https://www.rfc-editor.org/info/rfc4953>.

[RFC5927] Gont, F., "ICMP Attacks against TCP", RFC 5927, DOI 10.17487/RFC5927, July 2010, <https://www.rfc-editor.org/info/rfc5927>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

Authors' Addresses

Fernando Gont
SI6 Networks
Evaristo Carriego 2644
Haedo, Provincia de Buenos Aires 1706
Argentina

Phone: +54 11 4650 8472
Email: fgont@si6networks.com
URI: https://www.si6networks.com

Guillermo Gont
SI6 Networks
Evaristo Carriego 2644
Haedo, Provincia de Buenos Aires 1706
Argentina

Phone: +54 11 4650 8472
Email: ggont@si6networks.com
URI: https://www.si6networks.com

Miroslav Lichvar
Red Hat
Purkynova 115
Brno 612 00
Czech Republic

Email: mlichvar@redhat.com

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: August 25, 2021

A. Malhotra
Boston University
A. Langley
Google
W. Ladd
Cloudflare
M. Dansarie

February 21, 2021

Roughtime
draft-ietf-ntp-roughtime-04

Abstract

This document specifies Roughtime - a protocol that aims to achieve rough time synchronization while detecting servers that provide inaccurate time and providing cryptographic proof of their malfeasance.

Table of Contents

1. Introduction

Time synchronization is essential to Internet security as many security protocols and other applications require synchronization [RFC7384] [MCBG]. Unfortunately widely deployed protocols such as the Network Time Protocol (NTP) [RFC5905] lack essential security features, and even newer protocols like Network Time Security (NTS) [RFC8915] lack mechanisms to ensure that the servers behave

correctly. Authenticating time servers prevents network adversaries from modifying time packets, but an authenticated time server still has full control over the contents of the time packet and may go rogue. The Roughtime protocol provides cryptographic proof of malfeasance, enabling clients to detect and prove to a third party a server's attempts to influence the time a client computes.

Protocol	Authenticated Server	Server Malfeasance Evidence
NTP, Chronos	N	N
NTP-MD5	Y*	N
NTP-Autokey	Y**	N
NTS	Y	N
Roughtime	Y	Y

Security Properties of current protocols

Table 1

Y* For security issues with symmetric-key based NTP-MD5 authentication, please refer to RFC 8573 [RFC8573].

Y** For security issues with Autokey Public Key Authentication, refer to [Autokey].

- If a server's timestamps do not fit into the time context of other servers' responses, then a Roughtime client can cryptographically prove this misbehavior to third parties. This helps detect "bad" servers.

- A Roughtime client can roughly detect (with no absolute guarantee) a delay attack [DelayAttacks] but can not cryptographically prove this to a third party. However, the absence of proof of malfeasance should not be considered a proof of absence of malfeasance. So Roughtime should not be used as a witness that a server is overall "good".

- Note that delay attacks cannot be detected/stopped by any protocol. Delay attacks can not, however, undermine the security guarantees provided by Roughtime.

- Although delay attacks cannot be prevented, they can be limited to a predetermined upper bound. This can be done by defining a maximal tolerable Round Trip Time (RTT) value, MAX-RTT, that a Roughtime client is willing to accept. A Roughtime client can measure the RTT of every request-response handshake and compare it

to MAX-RTT. If the RTT exceeds MAX-RTT, the corresponding server is assumed to be a falseticker. When this approach is used the maximal time error that can be caused by a delay attack is MAX-RTT/2. It should be noted that this approach assumes that the nature of the system is known to the client, including reasonable upper bounds on the RTT value.

2. Requirements Language

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

3. Protocol Overview

Roughtime is a protocol for rough time synchronization that enables clients to provide cryptographic proof of server malfeasance. It does so by having responses from servers include a signature with a certificate rooted in a long-term public/private key pair over a value derived from a nonce provided by the client in its request. This provides cryptographic proof that the timestamp was issued after the server received the client's request. The derived value included in the server's response is the root of a Merkle tree which includes the hash of the client's nonce as the value of one of its leaf nodes. This enables the server to amortize the relatively costly signing operation over a number of client requests.

Single server mode: At its most basic level, Roughtime is a one round protocol in which a completely fresh client requests the current time and the server sends a signed response. The response includes a timestamp and a radius used to indicate the server's certainty about the reported time. For example, a radius of 1,000,000 microseconds means the server is absolutely confident that the true time is within one second of the reported time.

The server proves freshness of its response as follows: The client's request contains a nonce. The server incorporates the nonce into its signed response so that the client can verify the server's signatures covering the nonce issued by the client. Provided that the nonce has sufficient entropy, this proves that the signed response could only have been generated after the nonce.

4. The Guarantee

A Roughtime server guarantees that a response to a query sent at t_1, received at t_2, and with timestamp t_3 has been created between the transmission of the query and its reception. If t_3 is not within that interval, a server inconsistency may be detected and used to impeach the server. The propagation of such a guarantee and its use of type synchronization is discussed in Section 7. No delay attacker may affect this: they may only expand the interval between t_1 and t_2, or of course stop the measurement in the first place.

5. Message Format

Roughtime messages are maps consisting of one or more (tag, value) pairs. They start with a header, which contains the number of pairs, the tags, and value offsets. The header is followed by a message values section which contains the values associated with the tags in the header. Messages MUST be formatted according to Figure 1 as described in the following sections.

Messages may be recursive, i.e. the value of a tag can itself be a Roughtime message.

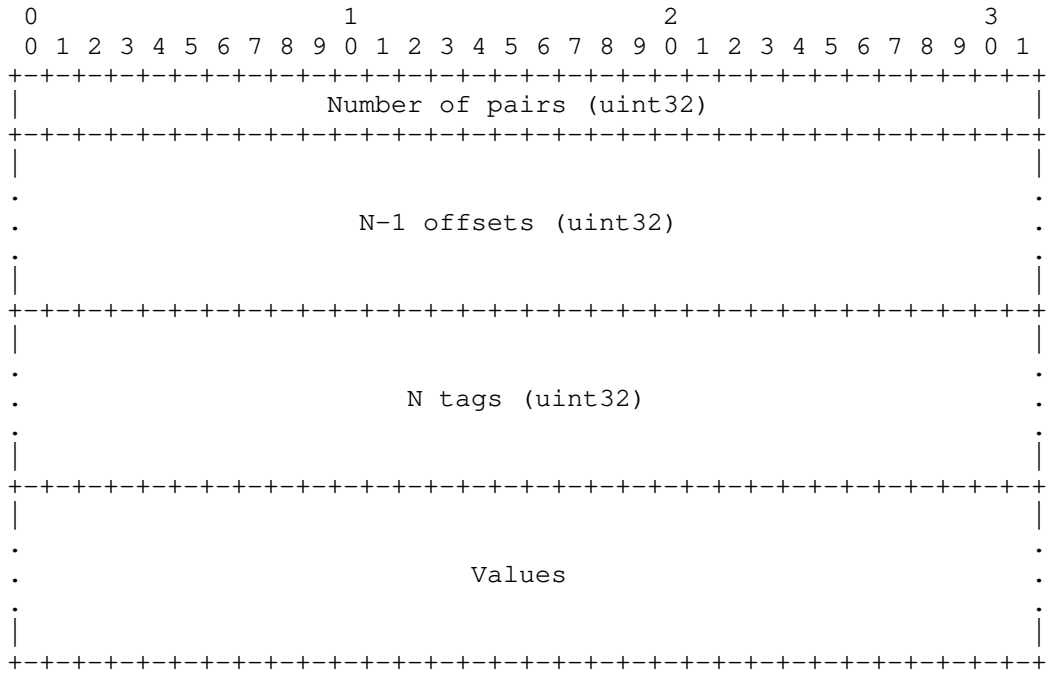

Figure 1: Roughtime Message Format

5.1. Data Types

5.1.1. int32

An int32 is a 32 bit signed integer. It is serialized in sign-magnitude representation with the sign bit in the most significant bit. It is serialized least significant byte first. The negative zero value (0x80000000) MUST NOT be used.

5.1.2. uint32

A uint32 is a 32 bit unsigned integer. It is serialized with the least significant byte first.

5.1.3. uint64

A uint64 is a 64 bit unsigned integer. It is serialized with the least significant byte first.

5.1.4. Tag

Tags are used to identify values in Roughtime messages. A tag is a uint32 but may also be listed as a sequence of up to four ASCII characters [RFC0020]. ASCII strings shorter than four characters can be unambiguously converted to tags by padding them with zero bytes. For example, the ASCII string "NONC" would correspond to the tag 0x434e4f4e and "PAD" would correspond to 0x00444150.

5.1.5. Timestamp

A timestamp is a uint64 interpreted in the following way. The most significant 3 bytes contain the integer part of a Modified Julian Date (MJD). The least significant 5 bytes is a count of the number of Coordinated Universal Time (UTC) microseconds [ITU-R_TF.460-6] since midnight on that day.

The MJD is the number of UTC days since 17 November 1858 [ITU-R_TF.457-2]. It is useful to note that 1 January 1970 is 40,587 days after 17 November 1858.

Note that, unlike NTP, this representation does not use the full number of bits in the fractional part and that days with leap seconds will have more or fewer than the nominal 86,400,000,000 microseconds.

5.2. Header

All Roughtime messages start with a header. The first four bytes of the header is the uint32 number of tags N, and hence of (tag, value) pairs. The following 4*(N-1) bytes are offsets, each a uint32. The last 4*N bytes in the header are tags.

Offsets refer to the positions of the values in the message values section. All offsets MUST be multiples of four and placed in increasing order. The first post-header byte is at offset 0. The offset array is considered to have a not explicitly encoded value of 0 as its zeroth entry. The value associated with the ith tag begins at offset[i] and ends at offset[i+1]-1, with the exception of the last value which ends at the end of the message. Values may have zero length.

Tags MUST be listed in the same order as the offsets of their values. A tag MUST NOT appear more than once in a header. Tags MUST also be sorted in ascending order by numeric value.

6. Protocol

As described in Section 3, clients initiate time synchronization by sending requests containing a nonce to servers who send signed time responses in return. Roughtime packets can be sent between clients and servers either as UDP datagrams or via TCP streams. Servers SHOULD support the UDP transport mode, while TCP transport is OPTIONAL.

A Roughtime packet MUST be formatted according to Figure 2 and as described here. The first field is a uint64 with the value 0x4d49544847554f52 ("ROUGHTIM" in ASCII). The second field is a uint32 and contains the length of the third field. The third and last field contains a Roughtime message as specified in Section 5.1.

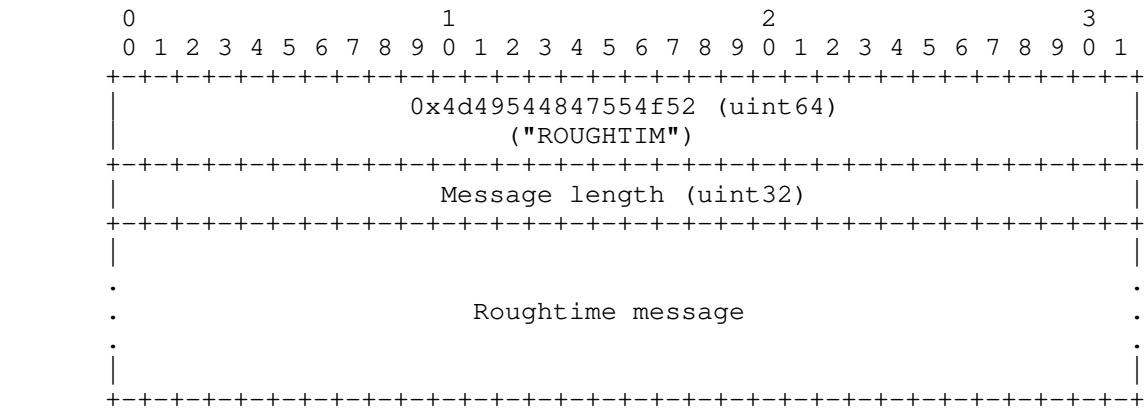

Figure 2: Roughtime Packet Format

Roughtime request and response packets MUST be transmitted in a single datagram when the UDP transport mode is used. Setting the packet's don't fragment bit [RFC0791] is OPTIONAL in IPv4 networks.

Multiple requests and responses can be exchanged over an established TCP connection. Clients MAY send multiple requests at once and servers MAY send responses out of order. The connection SHOULD be closed by the client when it has no more requests to send and has received all expected responses. Either side SHOULD close the connection in response to synchronization, format, implementation-defined timeouts, or other errors.

All requests and responses MUST contain the VER tag. It contains a list of one or more uint32 version numbers. The version of Roughtime specified by this memo has version number 1.

For testing drafts of this memo, a version number of 0x80000000 plus the draft number is used.

6.1. Requests

A request MUST contain the tags NONC and VER.

The value of the NONC tag is a 64 byte nonce. It SHOULD be generated in a manner indistinguishable from random.

In a request, the VER tag contains a list of versions. The VER tag MUST include at least one Roughtime version supported by the client. The client MUST ensure that the version numbers and tags included in the request are not incompatible with each other or the packet contents.

Tags other than NONC and VER SHOULD be ignored by the server.

The size of the request message SHOULD be at least 1024 bytes when the UDP transport mode is used. To attain this size the PAD tag SHOULD be added to the message. Its value SHOULD be all zeros. Responding to requests shorter than 1024 bytes is OPTIONAL and servers MUST NOT send responses larger than the requests they are replying to.

6.2. Responses

A response MUST contain the tags CERT, INDX, NONC, PATH, SIG, SREP, and VER.

The SIG tag is a signature over the SREP value using the public key contained in CERT, as explained below.

The SREP tag contains a time response. Its value is a Roughtime message with the tags ROOT, MIDP, and RADI. The server MAY include any of the tags DUT1, DTAI and LEAP in the contents of the SREP tag.

The NONC tag contains the nonce of the message being responded to.

The ROOT tag contains a 32 byte value of a Merkle tree root as described in Section 6.3.

The MIDP tag value is a timestamp of the moment of processing.

The RADI tag value is a uint32 representing the server's estimate of the accuracy of MIDP in microseconds. Servers MUST ensure that the true time is within (MIDP-RADI, MIDP+RADI) at the time they compose the response message.

The DUT1 tag value is an int32 indicating the predicted difference between UT1 and UTC (UT1 - UTC) in milliseconds as given by the International Earth Rotation and Reference Systems Service (IERS).

The DTAI tag value is an int32 indicating the current difference between International Atomic Time (TAI) and UTC (TAI - UTC) in milliseconds as published in the International Bureau of Weights and Measures' (BIPM) Circular T.

The LEAP tag contains zero or more int32 values, each representing a past or future leap second event. Positive values represent the addition of a second and negative values represent the removal of a second. The absolute value represents the MJD of the second after the leap second event, i.e., the first second with a new UTC - TAI difference. The leap second events MUST be sorted in reverse

chronological order and the first item MUST be the last (past or future) leap second event that the server knows about. A LEAP tag with zero int32 values indicates that the server does not hold any updated leap second information.

The SIG tag value is a 64 byte Ed25519 signature [RFC8032] over a signature context concatenated with the entire value of a DELE or SREP tag. Signatures of DELE tags MUST use the ASCII string "RoughTime v1 delegation signature--" and signatures of SREP tags MUST use the ASCII string "RoughTime v1 response signature" as signature context. Both strings MUST include a terminating zero byte.

The CERT tag contains a public-key certificate signed with the server's long-term key. Its value is a Roughtime message with the tags DELE and SIG, where SIG is a signature over the DELE value.

The DELE tag contains a delegated public-key certificate used by the server to sign the SREP tag. Its value is a Roughtime message with the tags MINT, MAXT, and PUBK. The purpose of the DELE tag is to enable separation of a long-term public key from keys on devices exposed to the public Internet.

The MINT tag is the minimum timestamp for which the key in PUBK is trusted to sign responses. MIDP MUST be more than or equal to MINT for a response to be considered valid.

The MAXT tag is the maximum timestamp for which the key in PUBK is trusted to sign responses. MIDP MUST be less than or equal to MAXT for a response to be considered valid.

The PUBK tag contains a temporary 32 byte Ed25519 public key which is used to sign the SREP tag.

The INDX tag value is a uint32 determining the position of NONC in the Merkle tree used to generate the ROOT value as described in Section 6.3.

The PATH tag value is a multiple of 32 bytes long and represents a path of 32 byte hash values in the Merkle tree used to generate the ROOT value as described in Section 6.3. In the case where a response is prepared for a single request and the Merkle tree contains only the root node, the size of PATH is zero.

In a response, the VER tag MUST contain a single version number. It SHOULD be one of the version numbers supplied by the client in its request. The server MUST ensure that the version number corresponds with the rest of the packet contents.

6.3. The Merkle Tree

A Merkle tree is a binary tree where the value of each non-leaf node is a hash value derived from its two children. The root of the tree is thus dependent on all leaf nodes.

In Roughtime, each leaf node in the Merkle tree represents the nonce of one request that a response message is sent in reply to. Leaf nodes are indexed left to right, beginning with zero.

The values of all nodes are calculated from the leaf nodes and up towards the root node using the first 32 bytes of the output of the SHA-512 hash algorithm [SHS]. For leaf nodes, the byte 0x00 is prepended to the nonce before applying the hash function. For all other nodes, the byte 0x01 is concatenated with first the left and then the right child node value before applying the hash function.

The value of the Merkle tree's root node is included in the ROOT tag of the response.

The index of a request's nonce node is included in the INDX tag of the response.

The values of all sibling nodes in the path between a request's nonce node and the root node is stored in the PATH tag so that the client can reconstruct and validate the value in the ROOT tag using its nonce.

6.3.1. Root Value Validity Check Algorithm

One starts by computing the hash of the NONC value from the request, with 0x00 prepended. Then one walks from the least significant bit of INDX to the most significant bit, and also walks towards the end of PATH.

If PATH ends then the remaining bits of the INDX MUST be all zero. This indicates the termination of the walk, and the current value MUST equal ROOT if the response is valid.

If the current bit is 0, one hashes 0x01, the current hash, and the value from PATH to derive the next current value.

If the current bit is 1 one hashes 0x01, the value from PATH, and the current hash to derive the next current value.

6.4. Validity of Response

A client MUST check the following properties when it receives a response. We assume the long-term server public key is known to the client through other means.

- The signature in CERT was made with the long-term key of the server.
- The DELE timestamps and the MIDP value are consistent.
- The INDX and PATH values prove NONC was included in the Merkle tree with value ROOT using the algorithm in Section 6.3.1.
- The signature of SREP in SIG validates with the public key in DELE.

A response that passes these checks is said to be valid. Validity of a response does not prove the time is correct, but merely that the server signed it, and thus guarantees that it began to compute the signature at a time in the interval (MIDP-RADI, MIDP+RADI).

7. Integration into NTP

We assume that there is a bound PHI on the frequency error in the clock on the machine. Given a measurement taken at a local time t1, we know the true time is in [t1-delta-sigma, t1-delta+sigma]. After d seconds have elapsed we know the true time is within [t1-delta-sigma-d*PHI, t1-delta+sigma+d*PHI]. A simple and effective way to mix with NTP or PTP discipline of the clock is to trim the observed intervals in NTP to fit entirely within this window or reject measurements that fall to far outside. This assumes time has not been stepped. If the NTP process decides to step the time, it MUST use Roughtime to ensure the new truetime estimate that will be stepped to is consistent with the true time.

Should this window become too large, another Roughtime measurement is called for. The definition of "too large" is implementation defined.

Implementations MAY use other, more sophisticated means of adjusting the clock respecting Roughtime information.

8. Grease

Servers MAY send back a fraction of responses that are syntactically invalid or contain invalid signatures as well as incorrect times. Clients MUST properly reject such responses. Servers MUST NOT send

back responses with incorrect times and valid signatures. Either signature MAY be invalid for this application.

9. Roughtime Servers

The below list contains a list of servers with their public keys in Base64 format. These servers may implement older versions of this specification.

```
address:       roughtime.cloudflare.com
port:          2002
long-term key: gD63hSj3ScS+wuOeGrubXlq35N1c5Lby/S+T7MNTjxo=

address:       roughtime.int08h.com
port:          2002
long-term key: AW5uAoTSTDfG5NfY1bTh08GUnOqlRb+HVhbJ3ODJvsE=

address:       roughtime.sandbox.google.com
port:          2002
long-term key: etPaaIxcBMY1oUeGpwvPMCJMwlRVNxv51KK/tktoJTQ=

address:       roughtime.se
port:          2002
long-term key: S3AzfZJ5CjSdkJ21ZJGbxqdYP/SoE8fXKY0+aicsehI=
```

10. Acknowledgements

Thomas Peterson corrected multiple nits. Peter Loethberg (Lothberg), Tal Mizrahi, Ragnar Sundblad, Kristof Teichel, and the other members of the NTP working group contributed comments and suggestions.

11. IANA Considerations

11.1. Service Name and Transport Protocol Port Number Registry

IANA is requested to allocate the following entry in the Service Name and Transport Protocol Port Number Registry [RFC6335]:

Service Name: Roughtime

Transport Protocol: tcp,udp

Assignee: IESG <iesg@ietf.org>

Contact: IETF Chair <chair@ietf.org>

Description: Roughtime time synchronization

Reference: [[this memo]]

Port Number: [[TBD1]], selected by IANA from the User Port range

11.2. Roughtime Version Registry

IANA is requested to create a new registry entitled " Roughtime Version Registry " Entries shall have the following fields:

Version ID (REQUIRED): a 32-bit unsigned integer

Version name (REQUIRED): A short text string naming the version being identified.

Reference (REQUIRED): A reference to a relevant specification document.

The policy for allocation of new entries SHOULD be: IETF Review.

The initial contents of this registry shall be as follows:

Version ID	Version name	Reference
0x0	Reserved	[[this memo]]
0x1	Roughtime version 1	[[this memo]]
0x2-0x7fffffff	Unassigned	
0x80000000-0xffffffff	Reserved for Private or Experimental use	[[this memo]]

11.3. Roughtime Tag Registry

IANA is requested to create a new registry entitled "Roughtime Tag Registry". Entries SHALL have the following fields:

Tag (REQUIRED): A 32-bit unsigned integer in hexadecimal format.

ASCII Representation (OPTIONAL): The ASCII representation of the tag in accordance with Section 5.1.4 of this memo, if applicable.

Reference (REQUIRED): A reference to a relevant specification document.

The policy for allocation of new entries in this registry SHOULD be: Specification Required.

The initial contents of this registry SHALL be as follows:

Tag	ASCII Representation	Reference
0x00444150	PAD	[[this memo]]
0x00474953	SIG	[[this memo]]
0x00524556	VER	[[this memo]]
0x31545544	DUT1	[[this memo]]
0x434e4f4e	NONC	[[this memo]]
0x454c4544	DELE	[[this memo]]
0x48544150	PATH	[[this memo]]
0x49415444	DTAI	[[this memo]]
0x49444152	RADI	[[this memo]]
0x4b425550	PUBK	[[this memo]]
0x5041454c	LEAP	[[this memo]]
0x5044494d	MIDP	[[this memo]]
0x50455253	SREP	[[this memo]]
0x544e494d	MINT	[[this memo]]
0x544f4f52	ROOT	[[this memo]]
0x54524543	CERT	[[this memo]]
0x5458414d	MAXT	[[this memo]]
0x58444e49	INDX	[[this memo]]

12. Security Considerations

Since the only supported signature scheme, Ed25519, is not quantum resistant, the Roughtime version described in this memo will not survive the advent of quantum computers.

Maintaining a list of trusted servers and adjudicating violations of the rules by servers is not discussed in this document and is essential for security. Roughtime clients MUST update their view of which servers are trustworthy in order to benefit from the detection of misbehavior.

Validating timestamps made on different dates requires knowledge of leap seconds in order to calculate time intervals correctly.

Servers carry out a significant amount of computation in response to clients, and thus may experience vulnerability to denial of service attacks.

This protocol does not provide any confidentiality, and given the nature of timestamps such impact is minor.

The compromise of a PUBK's private key, even past MAXT, is a problem as the private key can be used to sign invalid times that are in the range MINT to MAXT, and thus violate the good behavior guarantee of the server.

Servers MUST NOT send response packets larger than the request packets sent by clients, in order to prevent amplification attacks.

13. Privacy Considerations

This protocol is designed to obscure all client identifiers. Servers necessarily have persistent long-term identities essential to enforcing correct behavior.

Generating nonces in a nonrandom manner can cause leaks of private data or enable tracking of clients as they move between networks.

14. References

14.1. Normative References

[ITU-R_TF.457-2]
ITU-R, "Use of the Modified Julian Date by the Standard-Frequency and Time-Signal Services", ITU-R Recommendation TF.457-2, October 1997.

[ITU-R_TF.460-6]
ITU-R, "Standard-Frequency and Time-Signal Emissions", ITU-R Recommendation TF.460-6, February 2002.

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

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <https://www.rfc-editor.org/info/rfc6335>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[SHS] National Institute of Standards and Technology, "Secure Hash Standard", DOI 10.6028/NIST.FIPS.180-4, FIPS 180-4, August 2015.

14.2. Informative References

[Autokey] Rottger, S., "Analysis of the NTP Autokey Procedures", 2012, <https://zero-entropy.de/autokey_analysis.pdf>.

[DelayAttacks] Mizrahi, T., "A Game Theoretic Analysis of Delay Attacks Against Time Synchronization Protocols", DOI 10.1109/ISPCS.2012.6336612, 2012, <https://ieeexplore.ieee.org/document/6336612>.

[MCBG] Malhotra, A., Cohen, I., Brakke, E., and S. Goldberg, "Attacking the Network Time Protocol", 2015, <https://eprint.iacr.org/2015/1020>.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

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

[RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, DOI 10.17487/RFC7384, October 2014, <https://www.rfc-editor.org/info/rfc7384>.

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

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

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

Appendix A. Terms and Abbreviations

ASCII American Standard Code for Information Interchange

IANA Internet Assigned Numbers Authority

JSON JavaScript Object Notation [RFC8259]

MJD Modified Julian Date

NTP Network Time Protocol [RFC5905]

NTS Network Time Security [RFC8915]

TAI International Atomic Time (Temps Atomique International) [ITU-R_TF.460-6]

TCP Transmission Control Protocol [RFC0793]

UDP User Datagram Protocol [RFC0768]

UT Universal Time [ITU-R_TF.460-6]

UTC Coordinated Universal Time [ITU-R_TF.460-6]

Authors' Addresses

Aanchal Malhotra
Boston University
111 Cummington Mall
Boston 02215
USA

Email: aanchal4@bu.edu

Adam Langley
Google

Email:
agl@google.com

Watson Ladd
Cloudflare
101 Townsend St
San Francisco
USA

Email: watsonbladd@gmail.com

Marcus Dansarie
Sweden

Email: marcus@dansarie.se
URI: https://orcid.org/0000-0001-9246-0263

NTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 26, 2020

D. Franke
Akamai
D. Sibold
K. Teichel
PTB
M. Dansarie

R. Sundblad
Netnod

March 25, 2020

Network Time Security for the Network Time Protocol
draft-ietf-ntp-using-nts-for-ntp-28

Abstract

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

NTS is structured as a suite of two loosely coupled sub-protocols. The first (NTS-KE) handles initial authentication and key establishment over TLS. The second handles encryption and authentication during NTP time synchronization via extension fields in the NTP packets, and holds all required state only on the client via opaque cookies.

Table of Contents

1. Introduction

This memo specifies Network Time Security (NTS), a cryptographic security mechanism for network time synchronization. A complete specification is provided for application of NTS to the client-server mode of the Network Time Protocol (NTP) [RFC5905].

1.1. Objectives

The objectives of NTS are as follows:

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

- Authentication: Implementations can cryptographically verify that any time synchronization packets are authentic, i.e., that they were produced by an identified party and have not been modified in transit.

- Confidentiality: Although basic time synchronization data is considered non-confidential and sent in the clear, NTS includes support for encrypting NTP extension fields.

- Replay prevention: Client implementations can detect when a received time synchronization packet is a replay of a previous packet.

- Request-response consistency: Client implementations can verify that a time synchronization packet received from a server was sent in response to a particular request from the client.

- Unlinkability: For mobile clients, NTS will not leak any information additional to NTP which would permit a passive adversary to determine that two packets sent over different networks came from the same client.

- Non-amplification: Implementations (especially server implementations) can avoid acting as distributed denial-of-service

(DDoS) amplifiers by never responding to a request with a packet larger than the request packet.

- Scalability: Server implementations can serve large numbers of clients without having to retain any client-specific state.

- Performance: NTS must not significantly degrade the quality of the time transfer. The encryption and authentication used when actually transferring time should be lightweight (see RFC 7384, Section 5.7 [RFC7384]).

1.2. Protocol Overview

The Network Time Protocol includes many different operating modes to support various network topologies (see RFC 5905, Section 3 [RFC5905]). In addition to its best-known and most-widely-used client-server mode, it also includes modes for synchronization between symmetric peers, a control mode for server monitoring and administration, and a broadcast mode. These various modes have differing and partly contradictory requirements for security and performance. Symmetric and control modes demand mutual authentication and mutual replay protection. Additionally, for certain message types control mode may require confidentiality as well as authentication. Client-server mode places more stringent requirements on resource utilization than other modes, because servers may have vast number of clients and be unable to afford to maintain per-client state. However, client-server mode also has more relaxed security needs, because only the client requires replay protection: it is harmless for stateless servers to process replayed packets. The security demands of symmetric and control modes, on the other hand, are in conflict with the resource-utilization demands of client-server mode: any scheme which provides replay protection inherently involves maintaining some state to keep track of what messages have already been seen.

This memo specifies NTS exclusively for the client-server mode of NTP. To this end, NTS is structured as a suite of two protocols:

The "NTS Extensions for NTPv4" define a collection of NTP extension fields for cryptographically securing NTPv4 using previously-established key material. They are suitable for securing client-server mode because the server can implement them without retaining per-client state. All state is kept by the client and provided to the server in the form of an encrypted cookie supplied with each request. On the other hand, the NTS Extension Fields are suitable *only* for client-server mode because only the client, and not the server, is protected from replay.

> The "NTS Key Establishment" protocol (NTS-KE) is a mechanism for establishing key material for use with the NTS Extension Fields for NTPv4. It uses TLS to establish keys, provide the client with an initial supply of cookies, and negotiate some additional protocol options. After this, the TLS channel is closed with no per-client state remaining on the server side.

The typical protocol flow is as follows: The client connects to an NTS-KE server on the NTS TCP port and the two parties perform a TLS handshake. Via the TLS channel, the parties negotiate some additional protocol parameters and the server sends the client a supply of cookies along with an address and port of an NTP server for which the cookies are valid. The parties use TLS key export [RFC5705] to extract key material which will be used in the next phase of the protocol. This negotiation takes only a single round trip, after which the server closes the connection and discards all associated state. At this point the NTS-KE phase of the protocol is complete. Ideally, the client never needs to connect to the NTS-KE server again.

Time synchronization proceeds with the indicated NTP server. The client sends the server an NTP client packet which includes several extension fields. Included among these fields are a cookie (previously provided by the key establishment server) and an authentication tag, computed using key material extracted from the NTS-KE handshake. The NTP server uses the cookie to recover this key material and send back an authenticated response. The response includes a fresh, encrypted cookie which the client then sends back in the clear in a subsequent request. (This constant refreshing of cookies is necessary in order to achieve NTS's unlinkability goal.)

Figure 1 provides an overview of the high-level interaction between the client, the NTS-KE server, and the NTP server. Note that the cookies' data format and the exchange of secrets between NTS-KE and NTP servers are not part of this specification and are implementation dependent. However, a suggested format for NTS cookies is provided in Section 6.

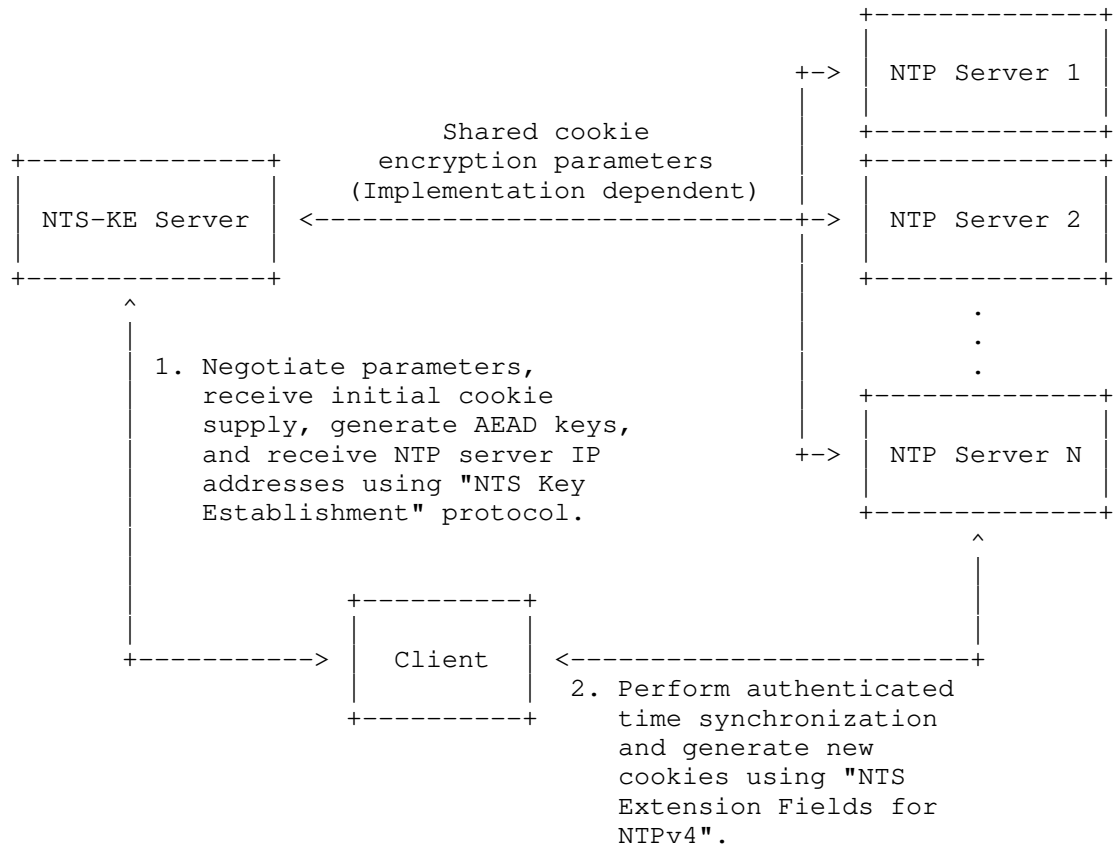

Figure 1: Overview of High-Level Interactions in NTS

2. Requirements Language

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

3. TLS profile for Network Time Security

Network Time Security makes use of TLS for NTS key establishment.

Since the NTS protocol is new as of this publication, no backward-compatibility concerns exist to justify using obsolete, insecure, or otherwise broken TLS features or versions. Implementations MUST conform with RFC 7525 [RFC7525] or with a later revision of BCP 195.

Implementations MUST NOT negotiate TLS versions earlier than 1.3 [RFC8446] and MAY refuse to negotiate any TLS version which has been superseded by a later supported version.

Use of the Application-Layer Protocol Negotiation Extension [RFC7301] is integral to NTS and support for it is REQUIRED for interoperability.

Implementations MUST follow the rules in RFC 5280 [RFC5280] and RFC 6125 [RFC6125] for the representation and verification of the application's service identity. When NTS-KE service discovery (out of scope for this document) produces one or more host names, use of the DNS-ID identifier type [RFC6125] is RECOMMENDED; specifications for service discovery mechanisms can provide additional guidance for certificate validation based on the results of discovery. Section 9.5 of this memo discusses particular considerations for certificate verification in the context of NTS.

4. The NTS Key Establishment Protocol

The NTS key establishment protocol is conducted via TCP port [[TBD1]]. The two endpoints carry out a TLS handshake in conformance with Section 3, with the client offering (via an ALPN [RFC7301] extension), and the server accepting, an application-layer protocol of "ntske/1". Immediately following a successful handshake, the client SHALL send a single request as Application Data encapsulated in the TLS-protected channel. Then, the server SHALL send a single response. After sending their respective request and response, the client and server SHALL send TLS "close_notify" alerts in accordance with RFC 8446, Section 6.1 [RFC8446].

The client's request and the server's response each SHALL consist of a sequence of records formatted according to Figure 2. The request and a non-error response each SHALL include exactly one NTS Next Protocol Negotiation record. The sequence SHALL be terminated by a "End of Message" record. The requirement that all NTS-KE messages be terminated by an End of Message record makes them self-delimiting.

Clients and servers MAY enforce length limits on requests and responses, however, servers MUST accept requests of at least 1024 octets and clients SHOULD accept responses of at least 65536 octets.

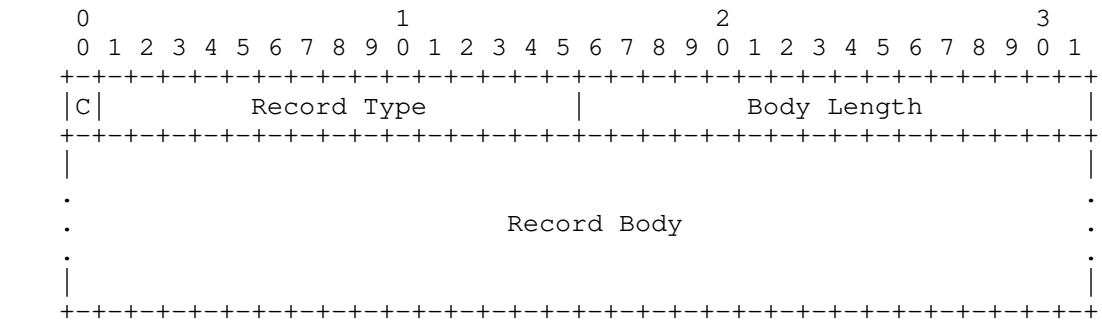

Figure 2: NTS-KE Record Format

The fields of an NTS-KE record are defined as follows:

C (Critical Bit): Determines the disposition of unrecognized Record Types. Implementations which receive a record with an unrecognized Record Type MUST ignore the record if the Critical Bit is 0 and MUST treat it as an error if the Critical Bit is 1 (see Section 4.1.3).

Record Type Number: A 15-bit integer in network byte order. The semantics of record types 0-7 are specified in this memo. Additional type numbers SHALL be tracked through the IANA Network Time Security Key Establishment Record Types registry.

Body Length: The length of the Record Body field, in octets, as a 16-bit integer in network byte order. Record bodies MAY have any representable length and need not be aligned to a word boundary.

Record Body: The syntax and semantics of this field SHALL be determined by the Record Type.

For clarity regarding bit-endianness: the Critical Bit is the most-significant bit of the first octet. In the C programming language, given a network buffer ‘unsigned char b[]‘ containing an NTS-KE record, the critical bit is ‘b[0] >> 7‘ while the record type is ‘((b[0] & 0x7f) << 8) + b[1]‘.

Note that, although the Type-Length-Body format of an NTS-KE record is similar to that of an NTP extension field, the semantics of the length field differ. While the length subfield of an NTP extension field gives the length of the entire extension field including the type and length subfields, the length field of an NTS-KE record gives just the length of the body.

Figure 3 provides a schematic overview of the key establishment. It displays the protocol steps to be performed by the NTS client and server and record types to be exchanged.

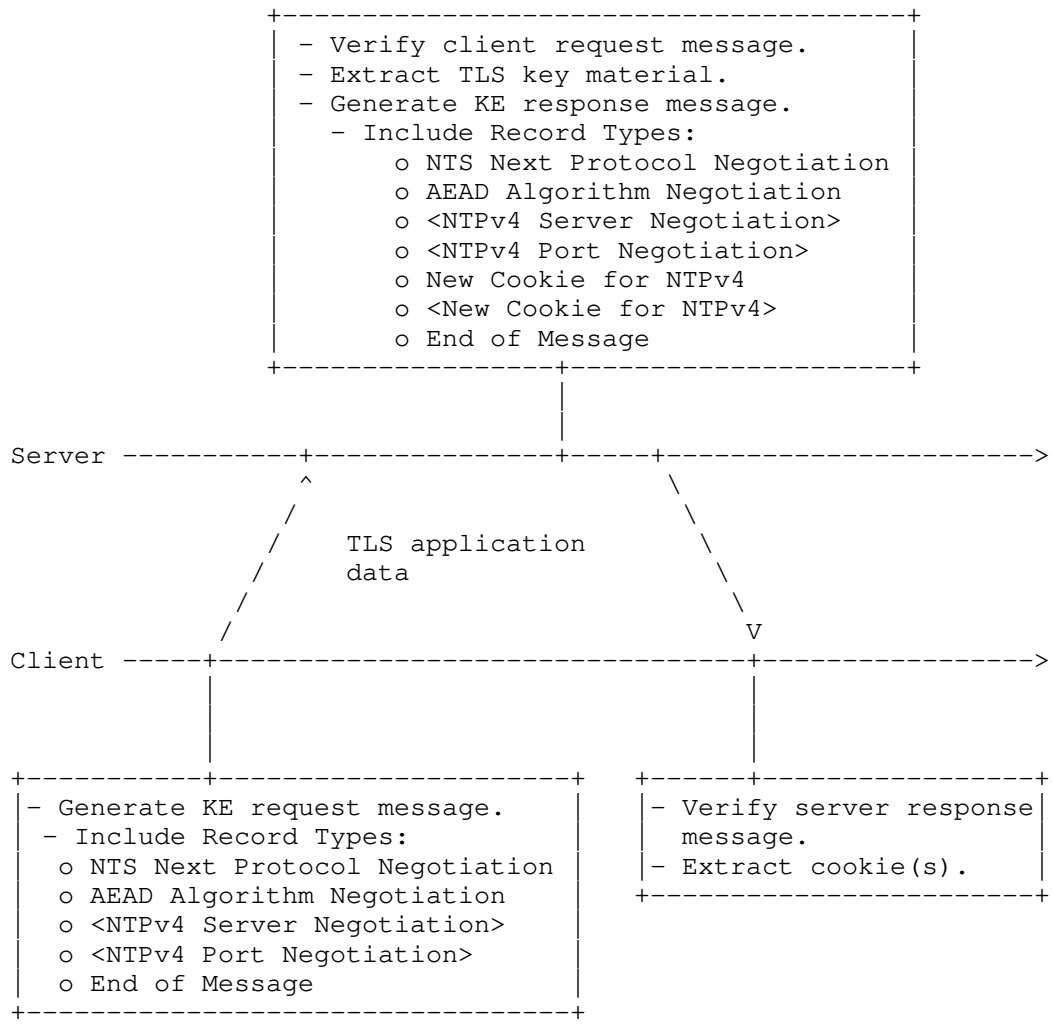

Figure 3: NTS Key Establishment Messages

4.1. NTS-KE Record Types

The following NTS-KE Record Types are defined:

4.1.1. End of Message

The End of Message record has a Record Type number of 0 and a zero-length body. It MUST occur exactly once as the final record of every NTS-KE request and response. The Critical Bit MUST be set.

4.1.2. NTS Next Protocol Negotiation

The NTS Next Protocol Negotiation record has a Record Type number of 1. It MUST occur exactly once in every NTS-KE request and response. Its body consists of a sequence of 16-bit unsigned integers in network byte order. Each integer represents a Protocol ID from the IANA Network Time Security Next Protocols registry. The Critical Bit MUST be set.

The Protocol IDs listed in the client's NTS Next Protocol Negotiation record denote those protocols which the client wishes to speak using the key material established through this NTS-KE session. Protocol IDs listed in the NTS-KE server's response MUST comprise a subset of those listed in the request and denote those protocols which the NTP server is willing and able to speak using the key material established through this NTS-KE session. The client MAY proceed with one or more of them. The request MUST list at least one protocol, but the response MAY be empty.

4.1.3. Error

The Error record has a Record Type number of 2. Its body is exactly two octets long, consisting of an unsigned 16-bit integer in network byte order, denoting an error code. The Critical Bit MUST be set.

Clients MUST NOT include Error records in their request. If clients receive a server response which includes an Error record, they MUST discard any key material negotiated during the initial TLS exchange and MUST NOT proceed to the Next Protocol. Requirements for retry intervals are described in Section 4.2.

The following error codes are defined:

> Error code 0 means "Unrecognized Critical Record". The server MUST respond with this error code if the request included a record which the server did not understand and which had its Critical Bit set. The client SHOULD NOT retry its request without modification.
>
> Error code 1 means "Bad Request". The server MUST respond with this error if the request is not complete and syntactically well-formed, or, upon the expiration of an implementation-defined

timeout, it has not yet received such a request. The client SHOULD NOT retry its request without modification.

Error code 2 means "Internal Server Error". The server MUST respond with this error if it is unable to respond properly due to an internal condition. The client MAY retry its request.

4.1.4. Warning

The Warning record has a Record Type number of 3. Its body is exactly two octets long, consisting of an unsigned 16-bit integer in network byte order, denoting a warning code. The Critical Bit MUST be set.

Clients MUST NOT include Warning records in their request. If clients receive a server response which includes a Warning record, they MAY discard any negotiated key material and abort without proceeding to the Next Protocol. Unrecognized warning codes MUST be treated as errors.

This memo defines no warning codes.

4.1.5. AEAD Algorithm Negotiation

The AEAD Algorithm Negotiation record has a Record Type number of 4. Its body consists of a sequence of unsigned 16-bit integers in network byte order, denoting Numeric Identifiers from the IANA AEAD Algorithms registry [IANA-AEAD]. The Critical Bit MAY be set.

If the NTS Next Protocol Negotiation record offers Protocol ID 0 (for NTPv4), then this record MUST be included exactly once. Other protocols MAY require it as well.

When included in a request, this record denotes which AEAD algorithms the client is willing to use to secure the Next Protocol, in decreasing preference order. When included in a response, this record denotes which algorithm the server chooses to use. It is empty if the server supports none of the algorithms offered. In requests, the list MUST include at least one algorithm. In responses, it MUST include at most one. Honoring the client's preference order is OPTIONAL: servers may select among any of the client's offered choices, even if they are able to support some other algorithm which the client prefers more.

Server implementations of NTS extension fields for NTPv4 (Section 5) MUST support AEAD_AES_SIV_CMAC_256 [RFC5297] (Numeric Identifier 15). That is, if the client includes AEAD_AES_SIV_CMAC_256 in its AEAD Algorithm Negotiation record and the server accepts Protocol ID 0

(NTPv4) in its NTS Next Protocol Negotiation record, then the server's AEAD Algorithm Negotiation record MUST NOT be empty.

4.1.6. New Cookie for NTPv4

The New Cookie for NTPv4 record has a Record Type number of 5. The contents of its body SHALL be implementation-defined and clients MUST NOT attempt to interpret them. See Section 6 for a suggested construction.

Clients MUST NOT send records of this type. Servers MUST send at least one record of this type, and SHOULD send eight of them, if the Next Protocol Negotiation response record contains Protocol ID 0 (NTPv4) and the AEAD Algorithm Negotiation response record is not empty. The Critical Bit SHOULD NOT be set.

4.1.7. NTPv4 Server Negotiation

The NTPv4 Server Negotiation record has a Record Type number of 6. Its body consists of an ASCII-encoded [RFC0020] string. The contents of the string SHALL be either an IPv4 address, an IPv6 address, or a fully qualified domain name (FQDN). IPv4 addresses MUST be in dotted decimal notation. IPv6 addresses MUST conform to the "Text Representation of Addresses" as specified in RFC 4291 [RFC4291] and MUST NOT include zone identifiers [RFC6874]. If a label contains at least one non-ASCII character, it is an internationalized domain name and an A-LABEL MUST be used as defined in Section 2.3.2.1 of RFC 5890 [RFC5890]. If the record contains a domain name, the recipient MUST treat it as a FQDN, e.g. by making sure it ends with a dot.

When NTPv4 is negotiated as a Next Protocol and this record is sent by the server, the body specifies the hostname or IP address of the NTPv4 server with which the client should associate and which will accept the supplied cookies. If no record of this type is sent, the client SHALL interpret this as a directive to associate with an NTPv4 server at the same IP address as the NTS-KE server. Servers MUST NOT send more than one record of this type.

When this record is sent by the client, it indicates that the client wishes to associate with the specified NTP server. The NTS-KE server MAY incorporate this request when deciding what NTPv4 Server Negotiation records to respond with, but honoring the client's preference is OPTIONAL. The client MUST NOT send more than one record of this type.

If the client has sent a record of this type, the NTS-KE server SHOULD reply with the same record if it is valid and the server is able to supply cookies for it. If the client has not sent any record

of this type, the NTS-KE server SHOULD respond with either an NTP server address in the same family as the NTS-KE session or a FQDN that can be resolved to an address in that family, if such alternatives are available.

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

4.1.8. NTPv4 Port Negotiation

The NTPv4 Port Negotiation record has a Record Type number of 7. Its body consists of a 16-bit unsigned integer in network byte order, denoting a UDP port number.

When NTPv4 is negotiated as a Next Protocol and this record is sent by the server, the body specifies the port number of the NTPv4 server with which the client should associate and which will accept the supplied cookies. If no record of this type is sent, the client SHALL assume a default of 123 (the registered port number for NTP).

When this record is sent by the client in conjunction with a NTPv4 Server Negotiation record, it indicates that the client wishes to associate with the NTP server at the specified port. The NTS-KE server MAY incorporate this request when deciding what NTPv4 Server Negotiation and NTPv4 Port Negotiation records to respond with, but honoring the client's preference is OPTIONAL.

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

4.2. Retry Intervals

A mechanism for not unnecessarily overloading the NTS-KE server is REQUIRED when retrying the key establishment process due to protocol, communication, or other errors. The exact workings of this will be dependent on the application and operational experience gathered over time. Until such experience is available, this memo provides the following suggestion.

Clients SHOULD use exponential backoff, with an initial and minimum retry interval of 10 seconds, a maximum retry interval of 5 days, and a base of 1.5. Thus, the minimum interval in seconds, 't', for the nth retry is calculated with

$$t = \min(10 * 1.5^{(n-1)}, 432000).$$

Clients MUST NOT reset the retry interval until they have performed a successful key establishment with the NTS-KE server, followed by a

successful use of the negotiated next protocol with the keys and data established during that transaction.

4.3. Key Extraction (generally)

Following a successful run of the NTS-KE protocol, key material SHALL be extracted using the HMAC-based Extract-and-Expand Key Derivation Function (HKDF) [RFC5869] in accordance with RFC 8446, Section 7.5 [RFC8446]. Inputs to the exporter function are to be constructed in a manner specific to the negotiated Next Protocol. However, all protocols which utilize NTS-KE MUST conform to the following two rules:

> The disambiguating label string [RFC5705] MUST be "EXPORTER-network-time-security".
>
> The per-association context value [RFC5705] MUST be provided and MUST begin with the two-octet Protocol ID which was negotiated as a Next Protocol.

5. NTS Extension Fields for NTPv4

5.1. Key Extraction (for NTPv4)

Following a successful run of the NTS-KE protocol wherein Protocol ID 0 (NTPv4) is selected as a Next Protocol, two AEAD keys SHALL be extracted: a client-to-server (C2S) key and a server-to-client (S2C) key. These keys SHALL be computed with the HKDF defined in RFC 8446, Section 7.5 [RFC8446] using the following inputs.

> The disambiguating label string [RFC5705] SHALL be "EXPORTER-network-time-security".
>
> The per-association context value [RFC5705] SHALL consist of the following five octets:
>
> > The first two octets SHALL be zero (the Protocol ID for NTPv4).
> >
> > The next two octets SHALL be the Numeric Identifier of the negotiated AEAD Algorithm in network byte order.
> >
> > The final octet SHALL be 0x00 for the C2S key and 0x01 for the S2C key.

Implementations wishing to derive additional keys for private or experimental use MUST NOT do so by extending the above-specified syntax for per-association context values. Instead, they SHOULD use their own disambiguating label string. Note that RFC 5705 [RFC5705]

provides that disambiguating label strings beginning with "EXPERIMENTAL" MAY be used without IANA registration.

5.2. Packet Structure Overview

In general, an NTS-protected NTPv4 packet consists of:

The usual 48-octet NTP header which is authenticated but not encrypted.

Some extension fields which are authenticated but not encrypted.

An extension field which contains AEAD output (i.e., an authentication tag and possible ciphertext). The corresponding plaintext, if non-empty, consists of some extension fields which benefit from both encryption and authentication.

Possibly, some additional extension fields which are neither encrypted nor authenticated. In general, these are discarded by the receiver.

Always included among the authenticated or authenticated-and-encrypted extension fields are a cookie extension field and a unique identifier extension field, as described in Section 5.7. The purpose of the cookie extension field is to enable the server to offload storage of session state onto the client. The purpose of the unique identifier extension field is to protect the client from replay attacks.

5.3. The Unique Identifier Extension Field

The Unique Identifier extension field provides the client with a cryptographically strong means of detecting replayed packets. It has a Field Type of [[TBD2]]. When the extension field is included in a client packet (mode 3), its body SHALL consist of a string of octets generated by a cryptographically secure random number generator [RFC4086]. The string MUST be at least 32 octets long. When the extension field is included in a server packet (mode 4), its body SHALL contain the same octet string as was provided in the client packet to which the server is responding. All server packets generated by NTS-implementing servers in response to client packets containing this extension field MUST also contain this field with the same content as in the client's request. The field's use in modes other than client-server is not defined.

This extension field MAY also be used standalone, without NTS, in which case it provides the client with a means of detecting spoofed packets from off-path attackers. Historically, NTP's origin

timestamp field has played both these roles, but for cryptographic purposes this is suboptimal because it is only 64 bits long and, depending on implementation details, most of those bits may be predictable. In contrast, the Unique Identifier extension field enables a degree of unpredictability and collision resistance more consistent with cryptographic best practice.

5.4. The NTS Cookie Extension Field

The NTS Cookie extension field has a Field Type of [[TBD3]]. Its purpose is to carry information which enables the server to recompute keys and other session state without having to store any per-client state. The contents of its body SHALL be implementation-defined and clients MUST NOT attempt to interpret them. See Section 6 for a suggested construction. The NTS Cookie extension field MUST NOT be included in NTP packets whose mode is other than 3 (client) or 4 (server).

5.5. The NTS Cookie Placeholder Extension Field

The NTS Cookie Placeholder extension field has a Field Type of [[TBD4]]. When this extension field is included in a client packet (mode 3), it communicates to the server that the client wishes it to send additional cookies in its response. This extension field MUST NOT be included in NTP packets whose mode is other than 3.

Whenever an NTS Cookie Placeholder extension field is present, it MUST be accompanied by an NTS Cookie extension field. The body length of the NTS Cookie Placeholder extension field MUST be the same as the body length of the NTS Cookie extension field. This length requirement serves to ensure that the response will not be larger than the request, in order to improve timekeeping precision and prevent DDoS amplification. The contents of the NTS Cookie Placeholder extension field's body SHOULD be all zeros and, aside from checking its length, MUST be ignored by the server.

5.6. The NTS Authenticator and Encrypted Extension Fields Extension Field

The NTS Authenticator and Encrypted Extension Fields extension field is the central cryptographic element of an NTS-protected NTP packet. Its Field Type is [[TBD5]]. It SHALL be formatted according to Figure 4 and include the following fields:

Nonce Length: Two octets in network byte order, giving the length of the Nonce field, excluding any padding, interpreted as an unsigned integer.

Ciphertext Length: Two octets in network byte order, giving the length of the Ciphertext field, excluding any padding, interpreted as an unsigned integer.

Nonce: A nonce as required by the negotiated AEAD Algorithm. The end of the field is zero-padded to a word (four octets) boundary.

Ciphertext: The output of the negotiated AEAD Algorithm. The structure of this field is determined by the negotiated algorithm, but it typically contains an authentication tag in addition to the actual ciphertext. The end of the field is zero-padded to a word (four octets) boundary.

Additional Padding: Clients which use a nonce length shorter than the maximum allowed by the negotiated AEAD algorithm may be required to include additional zero-padding. The necessary length of this field is specified below.

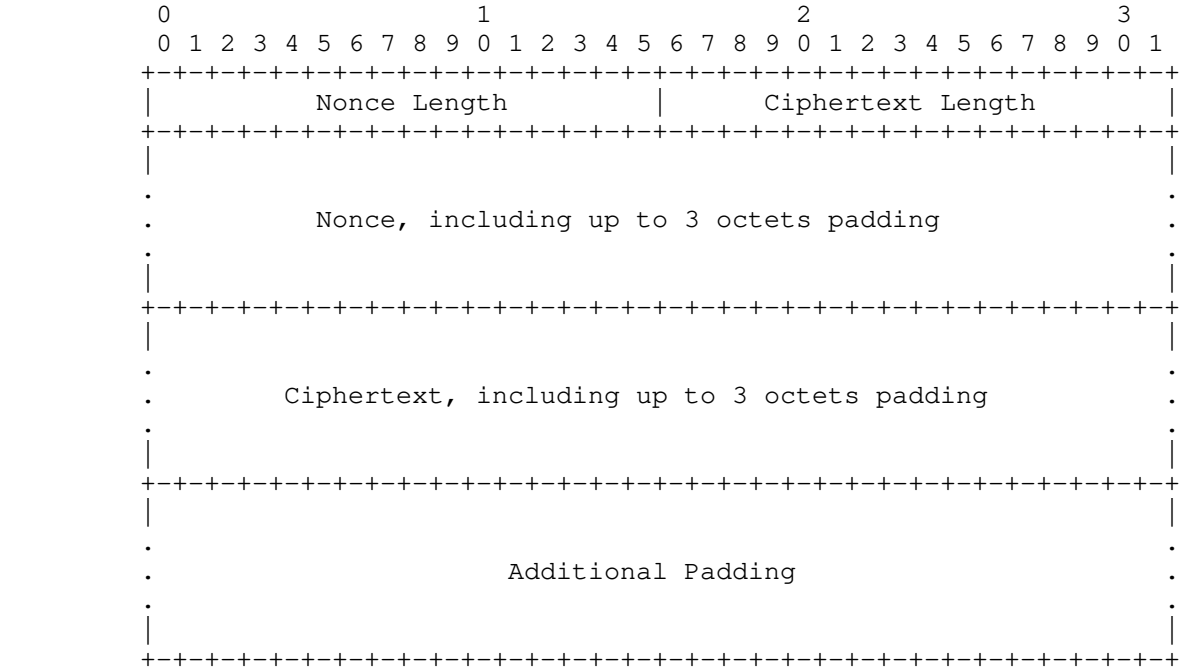

Figure 4: NTS Authenticator and Encrypted Extension Fields Extension Field Format

The Ciphertext field SHALL be formed by providing the following inputs to the negotiated AEAD Algorithm:

K: For packets sent from the client to the server, the C2S key SHALL be used. For packets sent from the server to the client, the S2C key SHALL be used.

A: The associated data SHALL consist of the portion of the NTP packet beginning from the start of the NTP header and ending at the end of the last extension field which precedes the NTS Authenticator and Encrypted Extension Fields extension field.

P: The plaintext SHALL consist of all (if any) NTP extension fields to be encrypted; if multiple extension fields are present they SHALL be joined by concatenation. Each such field SHALL be formatted in accordance with RFC 7822 [RFC7822], except that, contrary to the RFC 7822 requirement that fields have a minimum length of 16 or 28 octets, encrypted extension fields MAY be arbitrarily short (but still MUST be a multiple of 4 octets in length).

N: The nonce SHALL be formed however required by the negotiated AEAD algorithm.

The purpose of the Additional Padding field is to ensure that servers can always choose a nonce whose length is adequate to ensure its uniqueness, even if the client chooses a shorter one, and still ensure that the overall length of the server's response packet does not exceed the length of the request. For mode 4 (server) packets, no Additional Padding field is ever required. For mode 3 (client) packets, the length of the Additional Padding field SHALL be computed as follows. Let 'N_LEN' be the padded length of the Nonce field. Let 'N_MAX' be, as specified by RFC 5116 [RFC5116], the maximum permitted nonce length for the negotiated AEAD algorithm. Let 'N_REQ' be the lesser of 16 and N_MAX, rounded up to the nearest multiple of 4. If N_LEN is greater than or equal to N_REQ, then no Additional Padding field is required. Otherwise, the Additional Padding field SHALL be at least N_REQ - N_LEN octets in length. Servers MUST enforce this requirement by discarding any packet which does not conform to it.

Senders are always free to include more Additional Padding than mandated by the above paragraph. Theoretically, it could be necessary to do so in order to bring the extension field to the minimum length required by RFC 7822 [RFC7822]. This should never happen in practice because any reasonable AEAD algorithm will have a nonce and an authenticator long enough to bring the extension field to its required length already. Nonetheless, implementers are advised to explicitly handle this case and ensure that the extension field they emit is of legal length.

The NTS Authenticator and Encrypted Extension Fields extension field MUST NOT be included in NTP packets whose mode is other than 3 (client) or 4 (server).

5.7. Protocol Details

A client sending an NTS-protected request SHALL include the following extension fields as displayed in Figure 5:

Exactly one Unique Identifier extension field which MUST be authenticated, MUST NOT be encrypted, and whose contents MUST be the output of a cryptographically secure random number generator. [RFC4086]

Exactly one NTS Cookie extension field which MUST be authenticated and MUST NOT be encrypted. The cookie MUST be one which has been previously provided to the client, either from the key establishment server during the NTS-KE handshake or from the NTP server in response to a previous NTS-protected NTP request.

Exactly one NTS Authenticator and Encrypted Extension Fields extension field, generated using an AEAD Algorithm and C2S key established through NTS-KE.

To protect the client's privacy, the client SHOULD avoid reusing a cookie. If the client does not have any cookies that it has not already sent, it SHOULD initiate a re-run of the NTS-KE protocol. The client MAY reuse cookies in order to prioritize resilience over unlinkability. Which of the two that should be prioritized in any particular case is dependent on the application and the user's preference. Section 10.1 describes the privacy considerations of this in further detail.

The client MAY include one or more NTS Cookie Placeholder extension fields which MUST be authenticated and MAY be encrypted. The number of NTS Cookie Placeholder extension fields that the client includes SHOULD be such that if the client includes N placeholders and the server sends back N+1 cookies, the number of unused cookies stored by the client will come to eight. The client SHOULD NOT include more than seven NTS Cookie Placeholder extension fields in a request. When both the client and server adhere to all cookie-management guidance provided in this memo, the number of placeholder extension fields will equal the number of dropped packets since the last successful volley.

In rare circumstances, it may be necessary to include fewer NTS Cookie Placeholder extensions than recommended above in order to prevent datagram fragmentation. When cookies adhere the format

recommended in Section 6 and the AEAD in use is the mandatory-to-implement AEAD_AES_SIV_CMAC_256, senders can include a cookie and seven placeholders and still have packet size fall comfortably below 1280 octets if no non-NTS-related extensions are used; 1280 octets is the minimum prescribed MTU for IPv6 and is generally safe for avoiding IPv4 fragmentation. Nonetheless, senders SHOULD include fewer cookies and placeholders than otherwise indicated if doing so is necessary to prevent fragmentation.

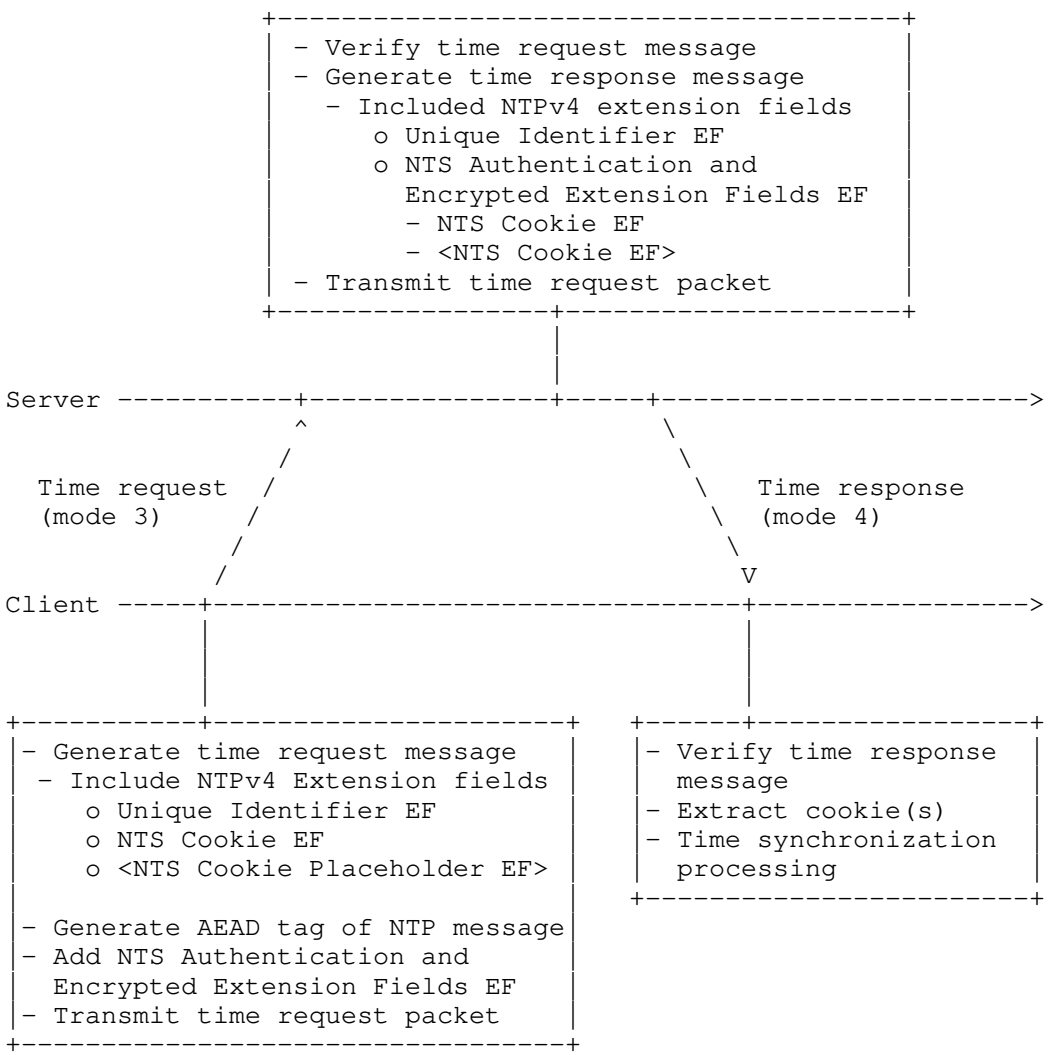

Figure 5: NTS-protected NTP Time Synchronization Messages

The client MAY include additional (non-NTS-related) extension fields which MAY appear prior to the NTS Authenticator and Encrypted Extension Fields extension fields (therefore authenticated but not encrypted), within it (therefore encrypted and authenticated), or after it (therefore neither encrypted nor authenticated). The server MUST discard any unauthenticated extension fields. Future specifications of extension fields MAY provide exceptions to this rule.

Upon receiving an NTS-protected request, the server SHALL (through some implementation-defined mechanism) use the cookie to recover the AEAD Algorithm, C2S key, and S2C key associated with the request, and then use the C2S key to authenticate the packet and decrypt the ciphertext. If the cookie is valid and authentication and decryption succeed, the server SHALL include the following extension fields in its response:

- Exactly one Unique Identifier extension field which MUST be authenticated, MUST NOT be encrypted, and whose contents SHALL echo those provided by the client.

- Exactly one NTS Authenticator and Encrypted Extension Fields extension field, generated using the AEAD algorithm and S2C key recovered from the cookie provided by the client.

- One or more NTS Cookie extension fields which MUST be authenticated and encrypted. The number of NTS Cookie extension fields included SHOULD be equal to, and MUST NOT exceed, one plus the number of valid NTS Cookie Placeholder extension fields included in the request. The cookies returned in those fields MUST be valid for use with the NTP server that sent them. They MAY be valid for other NTP servers as well, but there is no way for the server to indicate this.

We emphasize the contrast that NTS Cookie extension fields MUST NOT be encrypted when sent from client to server, but MUST be encrypted when sent from server to client. The former is necessary in order for the server to be able to recover the C2S and S2C keys, while the latter is necessary to satisfy the unlinkability goals discussed in Section 10.1. We emphasize also that "encrypted" means encapsulated within the NTS Authenticator and Encrypted Extensions extension field. While the body of an NTS Cookie extension field will generally consist of some sort of AEAD output (regardless of whether the recommendations of Section 6 are precisely followed), this is not sufficient to make the extension field "encrypted".

The server MAY include additional (non-NTS-related) extension fields which MAY appear prior to the NTS Authenticator and Encrypted

Extension Fields extension field (therefore authenticated but not encrypted), within it (therefore encrypted and authenticated), or after it (therefore neither encrypted nor authenticated). The client MUST discard any unauthenticated extension fields. Future specifications of extension fields MAY provide exceptions to this rule.

Upon receiving an NTS-protected response, the client MUST verify that the Unique Identifier matches that of an outstanding request, and that the packet is authentic under the S2C key associated with that request. If either of these checks fails, the packet MUST be discarded without further processing. In particular, the client MUST discard unprotected responses to NTS-protected requests.

If the server is unable to validate the cookie or authenticate the request, it SHOULD respond with a Kiss-o'-Death (KoD) packet (see RFC 5905, Section 7.4 [RFC5905]) with kiss code "NTSN", meaning "NTS NAK" (NTS negative-acknowledgment). It MUST NOT include any NTS Cookie or NTS Authenticator and Encrypted Extension Fields extension fields.

If the NTP server has previously responded with authentic NTS-protected NTP packets, the client MUST verify that any KoD packets received from the server contain the Unique Identifier extension field and that the Unique Identifier matches that of an outstanding request. If this check fails, the packet MUST be discarded without further processing. If this check passes, the client MUST comply with RFC 5905, Section 7.4 [RFC5905] where required.

A client MAY automatically re-run the NTS-KE protocol upon forced disassociation from an NTP server. In that case, it MUST avoid quickly looping between the NTS-KE and NTP servers by rate limiting the retries. Requirements for retry intervals in NTS-KE are described in Section 4.2.

Upon reception of the NTS NAK kiss code, the client SHOULD wait until the next poll for a valid NTS-protected response and if none is received, initiate a fresh NTS-KE handshake to try to renegotiate new cookies, AEAD keys, and parameters. If the NTS-KE handshake succeeds, the client MUST discard all old cookies and parameters and use the new ones instead. As long as the NTS-KE handshake has not succeeded, the client SHOULD continue polling the NTP server using the cookies and parameters it has.

To allow for NTP session restart when the NTS-KE server is unavailable and to reduce NTS-KE server load, the client SHOULD keep at least one unused but recent cookie, AEAD keys, negotiated AEAD algorithm, and other necessary parameters on persistent storage.

This way, the client is able to resume the NTP session without performing renewed NTS-KE negotiation.

6. Suggested Format for NTS Cookies

This section is non-normative. It gives a suggested way for servers to construct NTS cookies. All normative requirements are stated in Section 4.1.6 and Section 5.4.

The role of cookies in NTS is closely analogous to that of session cookies in TLS. Accordingly, the thematic resemblance of this section to RFC 5077 [RFC5077] is deliberate and the reader should likewise take heed of its security considerations.

Servers should select an AEAD algorithm which they will use to encrypt and authenticate cookies. The chosen algorithm should be one such as AEAD_AES_SIV_CMAC_256 [RFC5297] which resists accidental nonce reuse. It need not be the same as the one that was negotiated with the client. Servers should randomly generate and store a secret master AEAD key 'K'. Servers should additionally choose a non-secret, unique value 'I' as key-identifier for 'K'.

Servers should periodically (e.g., once daily) generate a new pair '(I,K)' and immediately switch to using these values for all newly-generated cookies. Following each such key rotation, servers should securely erase any previously generated keys that should now be expired. Servers should continue to accept any cookie generated using keys that they have not yet erased, even if those keys are no longer current. Erasing old keys provides for forward secrecy, limiting the scope of what old information can be stolen if a master key is somehow compromised. Holding on to a limited number of old keys allows clients to seamlessly transition from one generation to the next without having to perform a new NTS-KE handshake.

The need to keep keys synchronized between NTS-KE and NTP servers as well as across load-balanced clusters can make automatic key rotation challenging. However, the task can be accomplished without the need for central key-management infrastructure by using a ratchet, i.e., making each new key a deterministic, cryptographically pseudo-random function of its predecessor. A recommended concrete implementation of this approach is to use HKDF [RFC5869] to derive new keys, using the key's predecessor as Input Keying Material and its key identifier as a salt.

To form a cookie, servers should first form a plaintext 'P' consisting of the following fields:

The AEAD algorithm negotiated during NTS-KE.

The S2C key.

The C2S key.

Servers should then generate a nonce ‘N‘ uniformly at random, and form AEAD output ‘C‘ by encrypting ‘P‘ under key ‘K‘ with nonce ‘N‘ and no associated data.

The cookie should consist of the tuple ‘(I,N,C)‘.

To verify and decrypt a cookie provided by the client, first parse it into its components ‘I‘, ‘N‘, and ‘C‘. Use ‘I‘ to look up its decryption key ‘K‘. If the key whose identifier is ‘I‘ has been erased or never existed, decryption fails; reply with an NTS NAK. Otherwise, attempt to decrypt and verify ciphertext ‘C‘ using key ‘K‘ and nonce ‘N‘ with no associated data. If decryption or verification fails, reply with an NTS NAK. Otherwise, parse out the contents of the resulting plaintext ‘P‘ to obtain the negotiated AEAD algorithm, S2C key, and C2S key.

7. IANA Considerations

7.1. Service Name and Transport Protocol Port Number Registry

IANA is requested to allocate the following entry in the Service Name and Transport Protocol Port Number Registry [RFC6335]:

Service Name: ntske

Transport Protocol: tcp

Assignee: IESG <iesg@ietf.org>

Contact: IETF Chair <chair@ietf.org>

Description: Network Time Security Key Establishment

Reference: [[this memo]]

Port Number: [[TBD1]], selected by IANA from the User Port range

[[RFC EDITOR: Replace all instances of [[TBD1]] in this document with the IANA port assignment.]]

7.2. TLS Application-Layer Protocol Negotiation (ALPN) Protocol IDs Registry

IANA is requested to allocate the following entry in the TLS Application-Layer Protocol Negotiation (ALPN) Protocol IDs registry [RFC7301]:

Protocol: Network Time Security Key Establishment, version 1

Identification Sequence:
0x6E 0x74 0x73 0x6B 0x65 0x2F 0x31 ("ntske/1")

Reference: [[this memo]], Section 4

7.3. TLS Exporter Labels Registry

IANA is requested to allocate the following entry in the TLS Exporter Labels Registry [RFC5705]:

Value	DTLS-OK	Recommended	Reference	Note
EXPORTER-network-time-security	Y	Y	[[this memo]], Section 4.3	

7.4. NTP Kiss-o'-Death Codes Registry

IANA is requested to allocate the following entry in the registry of NTP Kiss-o'-Death Codes [RFC5905]:

Code	Meaning	Reference
NTSN	Network Time Security (NTS) negative-acknowledgment (NAK)	[[this memo]], Section 5.7

7.5. NTP Extension Field Types Registry

IANA is requested to allocate the following entries in the NTP Extension Field Types registry [RFC5905]:

Field Type	Meaning	Reference
[[TBD2]]	Unique Identifier	[[this memo]], Section 5.3
[[TBD3]]	NTS Cookie	[[this memo]], Section 5.4
[[TBD4]]	NTS Cookie Placeholder	[[this memo]], Section 5.5
[[TBD5]]	NTS Authenticator and Encrypted Extension Fields	[[this memo]], Section 5.6

[[RFC EDITOR: REMOVE BEFORE PUBLICATION - The NTP WG suggests that the following values be used:

Unique Identifier 0x0104
NTS Cookie 0x0204
Cookie Placeholder 0x0304
NTS Authenticator 0x0404]]

[[RFC EDITOR: Replace all instances of [[TBD2]], [[TBD3]], [[TBD4]], and [[TBD5]] in this document with the respective IANA assignments.]]

7.6. Network Time Security Key Establishment Record Types Registry

IANA is requested to create a new registry entitled "Network Time Security Key Establishment Record Types". Entries SHALL have the following fields:

Record Type Number (REQUIRED): An integer in the range 0-32767 inclusive.

Description (REQUIRED): A short text description of the purpose of the field.

Reference (REQUIRED): A reference to a document specifying the semantics of the record.

The policy for allocation of new entries in this registry SHALL vary by the Record Type Number, as follows:

0-1023: IETF Review

1024-16383: Specification Required

16384-32767: Private and Experimental Use

The initial contents of this registry SHALL be as follows:

Record Type Number	Description	Reference
0	End of Message	[[this memo]], Section 4.1.1
1	NTS Next Protocol Negotiation	[[this memo]], Section 4.1.2
2	Error	[[this memo]], Section 4.1.3
3	Warning	[[this memo]], Section 4.1.4
4	AEAD Algorithm Negotiation	[[this memo]], Section 4.1.5
5	New Cookie for NTPv4	[[this memo]], Section 4.1.6
6	NTPv4 Server Negotiation	[[this memo]], Section 4.1.7
7	NTPv4 Port Negotiation	[[this memo]], Section 4.1.8
16384-32767	Reserved for Private & Experimental Use	[[this memo]]

7.7. Network Time Security Next Protocols Registry

IANA is requested to create a new registry entitled "Network Time Security Next Protocols". Entries SHALL have the following fields:

Protocol ID (REQUIRED): An integer in the range 0-65535 inclusive, functioning as an identifier.

Protocol Name (REQUIRED): A short text string naming the protocol being identified.

Reference (REQUIRED): A reference to a relevant specification document.

The policy for allocation of new entries in these registries SHALL vary by their Protocol ID, as follows:

0-1023: IETF Review

1024-32767: Specification Required

32768-65535: Private and Experimental Use

The initial contents of this registry SHALL be as follows:

Protocol ID	Protocol Name	Reference
0	Network Time Protocol version 4 (NTPv4)	[[this memo]]
32768-65535	Reserved for Private or Experimental Use	Reserved by [[this memo]]

7.8. Network Time Security Error and Warning Codes Registries

IANA is requested to create two new registries entitled "Network Time Security Error Codes" and "Network Time Security Warning Codes". Entries in each SHALL have the following fields:

Number (REQUIRED): An integer in the range 0-65535 inclusive

Description (REQUIRED): A short text description of the condition.

Reference (REQUIRED): A reference to a relevant specification document.

The policy for allocation of new entries in these registries SHALL vary by their Number, as follows:

0-1023: IETF Review

1024-32767: Specification Required

32768-65535: Private and Experimental Use

The initial contents of the Network Time Security Error Codes Registry SHALL be as follows:

Number	Description	Reference
0	Unrecognized Critical Extension	[[this memo]], Section 4.1.3
1	Bad Request	[[this memo]], Section 4.1.3
2	Internal Server Error	[[this memo]], Section 4.1.3
32768-65535	Reserved for Private or Experimental Use	Reserved by [[this memo]]

The Network Time Security Warning Codes Registry SHALL initially be empty except for the reserved range, i.e.:

Number	Description	Reference
32768-65535	Reserved for Private or Experimental Use	Reserved by [[this memo]]

8. Implementation Status - RFC EDITOR: REMOVE BEFORE PUBLICATION

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

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

8.1. Implementation 1

Organization: Ostfalia University of Applied Science

Implementor: Martin Langer

Maturity: Proof-of-Concept Prototype

This implementation was used to verify consistency and to ensure completeness of this specification.

8.1.1. Coverage

This implementation covers the complete specification.

8.1.2. Licensing

The code is released under a Apache License 2.0 license.

The source code is available at: https://gitlab.com/MLanger/nts/

8.1.3. Contact Information

Contact Martin Langer: mart.langer@ostfalia.de

8.1.4. Last Update

The implementation was updated 25. February 2019.

8.2. Implementation 2

Organization: Netnod

Implementor: Christer Weinigel

Maturity: Proof-of-Concept Prototype

This implementation was used to verify consistency and to ensure completeness of this specification.

8.2.1. Coverage

This implementation covers the complete specification.

8.2.2. Licensing

The source code is available at: https://github.com/Netnod/nts-poc-python.

See LICENSE file for details on licensing (BSD 2).

8.2.3. Contact Information

Contact Christer Weinigel: christer@weinigel.se

8.2.4. Last Update

The implementation was updated 31. January 2019.

8.3. Implementation 3

Organization: Red Hat

Implementor: Miroslav Lichvar

Maturity: Prototype

This implementation was used to verify consistency and to ensure completeness of this specification.

8.3.1. Coverage

This implementation covers the complete specification.

8.3.2. Licensing

Licensing is GPLv2.

The source code is available at: https://github.com/mlichvar/chrony-nts

8.3.3. Contact Information

Contact Miroslav Lichvar: mlichvar@redhat.com

8.3.4. Last Update

The implementation was updated 28. March 2019.

8.4. Implementation 4

Organization: NTPsec

Implementor: Hal Murray and NTPsec team

Maturity:Looking for testers. Servers running at ntp1.glypnod.com:123 and ntp2.glypnod.com:123

This implementation was used to verify consistency and to ensure completeness of this specification.

8.4.1. Coverage

This implementation covers the complete specification.

8.4.2. Licensing

The source code is available at: https://gitlab.com/NTPsec/ntpsec. Licensing details in LICENSE.

8.4.3. Contact Information

Contact Hal Murray: hmurray@megapathdsl.net, devel@ntpsec.org

8.4.4. Last Update

The implementation was updated 2019-Apr-10.

8.5. Implementation 5

Organization: Cloudflare

Implementor: Watson Ladd

Maturity:

This implementation was used to verify consistency and to ensure completeness of this specification.

8.5.1. Coverage

This implementation covers the server side of the NTS specification.

8.5.2. Licensing

The source code is available at: https://github.com/wbl/nts-rust

Licensing is ISC (details see LICENSE.txt file).

8.5.3. Contact Information

Contact Watson Ladd: watson@cloudflare.com

8.5.4. Last Update

The implementation was updated 21. March 2019.

8.6. Implementation 6

Organization: Hacklunch, independent

Implementor: Michael Cardell Widerkrantz, Daniel Lublin, Martin Samuelsson et. al.

Maturity: interoperable client, immature server

8.6.1. Coverage

NTS-KE client and server.

8.6.2. Licensing

Licensing is ISC (details in LICENSE file).

Source code is available at: https://gitlab.com/hacklunch/ntsclient

8.6.3. Contact Information

Contact Michael Cardell Widerkrantz: mc@netnod.se

8.6.4. Last Update

The implementation was updated 6. February 2020.

8.7. Interoperability

The Interoperability tests distinguished between NTS key establishment protocol and NTS time exchange messages. For the implementations 1, 2, 3, and 4 pairwise interoperability of the NTS key establishment protocol and exchange of NTS protected NTP messages have been verified successfully. The implementation 2 was able to successfully perform the key establishment protocol against the server side of the implementation 5.

These tests successfully demonstrate that there are at least four running implementations of this draft which are able to interoperate.

9. Security Considerations

9.1. Protected Modes

NTP provides many different operating modes in order to support different network topologies and to adapt to various requirements. This memo only specifies NTS for NTP modes 3 (client) and 4 (server) (see Section 1.2). The best current practice for authenticating the other NTP modes is using the symmetric message authentication code feature as described in RFC 5905 [RFC5905] and RFC 8573 [RFC8573].

9.2. Cookie Encryption Key Compromise

If the suggested format for NTS cookies in Section 6 of this draft is used, an attacker who has gained access to the secret cookie encryption key 'K' can impersonate the NTP server, including generating new cookies. NTP and NTS-KE server operators SHOULD remove compromised keys as soon as the compromise is discovered. This will cause the NTP servers to respond with NTS NAK, thus forcing key renegotiation. Note that this measure does not protect against MITM attacks where the attacker has access to a compromised cookie encryption key. If another cookie scheme is used, there are likely similar considerations for that particular scheme.

9.3. Sensitivity to DDoS Attacks

The introduction of NTS brings with it the introduction of asymmetric cryptography to NTP. Asymmetric cryptography is necessary for initial server authentication and AEAD key extraction. Asymmetric cryptosystems are generally orders of magnitude slower than their symmetric counterparts. This makes it much harder to build systems that can serve requests at a rate corresponding to the full line speed of the network connection. This, in turn, opens up a new possibility for DDoS attacks on NTP services.

The main protection against these attacks in NTS lies in that the use of asymmetric cryptosystems is only necessary in the initial NTS-KE phase of the protocol. Since the protocol design enables separation of the NTS-KE and NTP servers, a successful DDoS attack on an NTS-KE server separated from the NTP service it supports will not affect NTP users that have already performed initial authentication, AEAD key extraction, and cookie exchange.

NTS users should also consider that they are not fully protected against DoS attacks by on-path adversaries. In addition to dropping packets and attacks such as those described in Section 9.6, an on-path attacker can send spoofed kiss-o'-death replies, which are not authenticated, in response to NTP requests. This could result in significantly increased load on the NTS-KE server. Implementers have to weigh the user's need for unlinkability against the added resilience that comes with cookie reuse in cases of NTS-KE server unavailability.

9.4. Avoiding DDoS Amplification

Certain non-standard and/or deprecated features of the Network Time Protocol enable clients to send a request to a server which causes the server to send a response much larger than the request. Servers which enable these features can be abused in order to amplify traffic

volume in DDoS attacks by sending them a request with a spoofed source IP. In recent years, attacks of this nature have become an endemic nuisance.

NTS is designed to avoid contributing any further to this problem by ensuring that NTS-related extension fields included in server responses will be the same size as the NTS-related extension fields sent by the client. In particular, this is why the client is required to send a separate and appropriately padded-out NTS Cookie Placeholder extension field for every cookie it wants to get back, rather than being permitted simply to specify a desired quantity.

Due to the RFC 7822 [RFC7822] requirement that extensions be padded and aligned to four-octet boundaries, response size may still in some cases exceed request size by up to three octets. This is sufficiently inconsequential that we have declined to address it.

9.5. Initial Verification of Server Certificates

NTS's security goals are undermined if the client fails to verify that the X.509 certificate chain presented by the NTS-KE server is valid and rooted in a trusted certificate authority. RFC 5280 [RFC5280] and RFC 6125 [RFC6125] specify how such verification is to be performed in general. However, the expectation that the client does not yet have a correctly-set system clock at the time of certificate verification presents difficulties with verifying that the certificate is within its validity period, i.e., that the current time lies between the times specified in the certificate's notBefore and notAfter fields. It may be operationally necessary in some cases for a client to accept a certificate which appears to be expired or not yet valid. While there is no perfect solution to this problem, there are several mitigations the client can implement to make it more difficult for an adversary to successfully present an expired certificate:

Check whether the system time is in fact unreliable. On systems with the ntp_adjtime() system call, a return code other than TIME_ERROR indicates that some trusted software has already set the time and certificates can be strictly validated.

Allow the system administrator to specify that certificates should *always* be strictly validated. Such a configuration is appropriate on systems which have a battery-backed clock and which can reasonably prompt the user to manually set an approximately-correct time if it appears to be needed.

Once the clock has been synchronized, periodically write the current system time to persistent storage. Do not accept any

certificate whose notAfter field is earlier than the last recorded time.

NTP time replies are expected to be consistent with the NTS-KE TLS certificate validity period, i.e. time replies received immediately after an NTS-KE handshake are expected to lie within the certificate validity period. Implementations are recommended to check that this is the case. Performing a new NTS-KE handshake based solely on the fact that the certificate used by the NTS-KE server in a previous handshake has expired is normally not necessary. Clients that still wish to do this must take care not to cause an inadvertent denial-of-service attack on the NTS-KE server, for example by picking a random time in the week preceding certificate expiry to perform the new handshake.

Use multiple time sources. The ability to pass off an expired certificate is only useful to an adversary who has compromised the corresponding private key. If the adversary has compromised only a minority of servers, NTP's selection algorithm (RFC 5905 section 11.2.1 [RFC5905]) will protect the client from accepting bad time from the adversary-controlled servers.

9.6. Delay Attacks

In a packet delay attack, an adversary with the ability to act as a man-in-the-middle delays time synchronization packets between client and server asymmetrically [RFC7384]. Since NTP's formula for computing time offset relies on the assumption that network latency is roughly symmetrical, this leads to the client to compute an inaccurate value [Mizrahi]. The delay attack does not reorder or modify the content of the exchanged synchronization packets. Therefore, cryptographic means do not provide a feasible way to mitigate this attack. However, the maximum error that an adversary can introduce is bounded by half of the round trip delay.

RFC 5905 [RFC5905] specifies a parameter called MAXDIST which denotes the maximum round-trip latency (including not only the immediate round trip between client and server, but the whole distance back to the reference clock as reported in the Root Delay field) that a client will tolerate before concluding that the server is unsuitable for synchronization. The standard value for MAXDIST is one second, although some implementations use larger values. Whatever value a client chooses, the maximum error which can be introduced by a delay attack is MAXDIST/2.

Usage of multiple time sources, or multiple network paths to a given time source [Shpiner], may also serve to mitigate delay attacks if the adversary is in control of only some of the paths.

9.7. NTS Stripping

Implementers must be aware of the possibility of "NTS stripping" attacks, where an attacker attempts to trick clients into reverting to plain NTP. Naive client implementations might, for example, revert automatically to plain NTP if the NTS-KE handshake fails. A man-in-the-middle attacker can easily cause this to happen. Even clients that already hold valid cookies can be vulnerable, since an attacker can force a client to repeat the NTS-KE handshake by sending faked NTP mode 4 replies with the NTS NAK kiss code. Forcing a client to repeat the NTS-KE handshake can also be the first step in more advanced attacks.

For the reasons described here, implementations SHOULD NOT revert from NTS-protected to unprotected NTP with any server without explicit user action.

10. Privacy Considerations

10.1. Unlinkability

Unlinkability prevents a device from being tracked when it changes network addresses (e.g. because said device moved between different networks). In other words, unlinkability thwarts an attacker that seeks to link a new network address used by a device with a network address that it was formerly using, because of recognizable data that the device persistently sends as part of an NTS-secured NTP association. This is the justification for continually supplying the client with fresh cookies, so that a cookie never represents recognizable data in the sense outlined above.

NTS's unlinkability objective is merely to not leak any additional data that could be used to link a device's network address. NTS does not rectify legacy linkability issues that are already present in NTP. Thus, a client that requires unlinkability must also minimize information transmitted in a client query (mode 3) packet as described in the draft [I-D.ietf-ntp-data-minimization].

The unlinkability objective only holds for time synchronization traffic, as opposed to key establishment traffic. This implies that it cannot be guaranteed for devices that function not only as time clients, but also as time servers (because the latter can be externally triggered to send linkable data, such as the TLS certificate).

It should also be noted that it could be possible to link devices that operate as time servers from their time synchronization traffic, using information exposed in (mode 4) server response packets (e.g.

reference ID, reference time, stratum, poll). Also, devices that respond to NTP control queries could be linked using the information revealed by control queries.

Note that the unlinkability objective does not prevent a client device to be tracked by its time servers.

10.2. Confidentiality

NTS does not protect the confidentiality of information in NTP's header fields. When clients implement [I-D.ietf-ntp-data-minimization], client packet headers do not contain any information which the client could conceivably wish to keep secret: one field is random, and all others are fixed. Information in server packet headers is likewise public: the origin timestamp is copied from the client's (random) transmit timestamp, and all other fields are set the same regardless of the identity of the client making the request.

Future extension fields could hypothetically contain sensitive information, in which case NTS provides a mechanism for encrypting them.

11. Acknowledgements

The authors would like to thank Richard Barnes, Steven Bellovin, Scott Fluhrer, Patrik Faeltstroem (Faltstrom), Sharon Goldberg, Russ Housley, Benjamin Kaduk, Suresh Krishnan, Mirja Kuehlewind (Kuehlewind), Martin Langer, Barry Leiba, Miroslav Lichvar, Aanchal Malhotra, Danny Mayer, Dave Mills, Sandra Murphy, Hal Murray, Karen O'Donoghue, Eric K. Rescorla, Kurt Roeckx, Stephen Roettger, Dan Romascanu, Kyle Rose, Rich Salz, Brian Sniffen, Susan Sons, Douglas Stebila, Harlan Stenn, Joachim Stroembergsson (Strombergsson), Martin Thomson, Eric (Eric) Vyncke, Richard Welty, Christer Weinigel, and Magnus Westerlund for contributions to this document and comments on the design of NTS.

12. References

12.1. Normative References

[IANA-AEAD]
IANA, "Authenticated Encryption with Associated Data (AEAD) Parameters", <https://www.iana.org/assignments/aead-parameters/>.

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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <https://www.rfc-editor.org/info/rfc4291>.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/info/rfc5116>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5297] Harkins, D., "Synthetic Initialization Vector (SIV) Authenticated Encryption Using the Advanced Encryption Standard (AES)", RFC 5297, DOI 10.17487/RFC5297, October 2008, <https://www.rfc-editor.org/info/rfc5297>.

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <https://www.rfc-editor.org/info/rfc5705>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <https://www.rfc-editor.org/info/rfc5890>.

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

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <https://www.rfc-editor.org/info/rfc6125>.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <https://www.rfc-editor.org/info/rfc6335>.

[RFC6874] Carpenter, B., Cheshire, S., and R. Hinden, "Representing IPv6 Zone Identifiers in Address Literals and Uniform Resource Identifiers", RFC 6874, DOI 10.17487/RFC6874, February 2013, <https://www.rfc-editor.org/info/rfc6874>.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <https://www.rfc-editor.org/info/rfc7301>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <https://www.rfc-editor.org/info/rfc7525>.

[RFC7822] Mizrahi, T. and D. Mayer, "Network Time Protocol Version 4 (NTPv4) Extension Fields", RFC 7822, DOI 10.17487/RFC7822, March 2016, <https://www.rfc-editor.org/info/rfc7822>.

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

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

12.2. Informative References

[I-D.ietf-ntp-data-minimization]
Franke, D. and A. Malhotra, "NTP Client Data Minimization", draft-ietf-ntp-data-minimization-04 (work in progress), March 2019.

[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, DOI 10.1109/ISPCS.2012.6336612, September 2012.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, DOI 10.17487/RFC5077, January 2008, <https://www.rfc-editor.org/info/rfc5077>.

[RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, DOI 10.17487/RFC7384, October 2014, <https://www.rfc-editor.org/info/rfc7384>.

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

[Shpiner] Shpiner, A., Revah, Y., and T. Mizrahi, "Multi-path Time Protocols", in Proceedings of IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS), DOI 10.1109/ISPCS.2013.6644754, September 2013.

Appendix A. Terms and Abbreviations

AEAD Authenticated Encryption with Associated Data [RFC5116]

ALPN Application-Layer Protocol Negotiation [RFC7301]

C2S Client-to-server

DoS Denial-of-Service

DDoS Distributed Denial-of-Service

EF Extension Field [RFC5905]

HKDF Hashed Message Authentication Code-based Key Derivation Function [RFC5869]

KoD	Kiss-o'-Death [RFC5905]
NTP	Network Time Protocol [RFC5905]
NTS	Network Time Security
NTS NAK	NTS negative-acknowledgment
NTS-KE	Network Time Security Key Establishment
S2C	Server-to-client
TLS	Transport Layer Security [RFC8446]

Authors' Addresses

Daniel Fox Franke
Akamai Technologies
145 Broadway
Cambridge, MA 02142
United States

Email: dafranke@akamai.com

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)531-592-4471
Email: kristof.teichel@ptb.de

Marcus Dansarie
Sweden

Email: marcus@dansarie.se
URI: https://orcid.org/0000-0001-9246-0263

Ragnar Sundblad
Netnod
Sweden

Email: ragge@netnod.se

NTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 26, 2021

N. Wu
D. Dhody, Ed.
Huawei
A. Sinha, Ed.
A. Kumar S N
RtBrick Inc.
Y. Zhao
Ericsson

February 22, 2021

A YANG Data Model for NTP
draft-ietf-ntp-yang-data-model-14

Abstract

This document defines a YANG data model for Network Time Protocol (NTP) 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", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Status of This Memo

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

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

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

This Internet-Draft will expire on August 26, 2021.

Table of Contents

1. Introduction

This document defines a YANG [RFC7950] data model for Network Time Protocol [RFC5905] implementations.

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

1.1. Operational State

NTP Operational State is included in the same tree as NTP configuration, consistent with Network Management Datastore Architecture (NMDA) [RFC8342]. NTP current state and statistics are also maintained in the operational state. The operational state also includes the NTP association state.

1.2. Terminology

The terminology used in this document is aligned to [RFC5905].

1.3. Tree Diagrams

A simplified graphical representation of the data model is used in this document. This document uses the graphical representation of data models defined in [RFC8340].

1.4. Prefixes in Data Node Names

In this document, names of data nodes and other data model objects are often used without a prefix, as long as it is clear from the context in which YANG module each name is defined. Otherwise, names are prefixed using the standard prefix associated with the corresponding YANG module, as shown in Table 1.

Prefix	YANG module	Reference
yang	ietf-yang-types	[RFC6991]
inet	ietf-inet-types	[RFC6991]
if	ietf-interfaces	[RFC8343]
sys	ietf-system	[RFC7317]
acl	ietf-access-control-list	[RFC8519]
rt-types	ietf-routing-types	[RFC8294]
nacm	ietf-netconf-acm	[RFC8341]

Table 1: Prefixes and corresponding YANG modules

1.5. References in the Model

Following documents are referenced in the model defined in this document –

Title	Reference
Network Time Protocol Version 4: Protocol and Algorithms Specification	[RFC5905]
Common YANG Data Types	[RFC6991]
A YANG Data Model for System Management	[RFC7317]
YANG Data Model for Key Chains	[RFC8177]
Common YANG Data Types for the Routing Area	[RFC8294]
Network Configuration Access Control Model	[RFC8341]
A YANG Data Model for Interface Management	[RFC8343]
YANG Data Model for Network Access Control Lists (ACLs)	[RFC8519]
Message Authentication Code for the Network Time Protocol	[RFC8573]
The AES-CMAC Algorithm	[RFC4493]
The MD5 Message-Digest Algorithm	[RFC1321]
US Secure Hash Algorithm 1 (SHA1)	[RFC3174]

Table 2: References in the YANG modules

2. NTP data model

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

```
module: ietf-ntp
  +--rw ntp!
```

```
      +--rw port?                    inet:port-number {ntp-port}?
      +--rw refclock-master!
      |  +--rw master-stratum?   ntp-stratum
      +--rw authentication {authentication}?
      |  +--rw auth-enabled?          boolean
      |  +--rw authentication-keys* [key-id]
      |     +--rw key-id       uint32
      |     |  ...
      +--rw access-rules {access-rules}?
      |  +--rw access-rule* [access-mode]
      |     +--rw access-mode    identityref
      |     +--rw acl?           -> /acl:acls/acl/name
      +--ro clock-state
      |  +--ro system-status
      |     +--ro clock-state                 identityref
      |     +--ro clock-stratum               ntp-stratum
      |     +--ro clock-refid                 refid
      |     |  ...
      +--rw unicast-configuration* [address type]
      |       {unicast-configuration}?
      |  +--rw address           inet:ip-address
      |  +--rw type              identityref
      |  |  ...
      +--ro associations* [address local-mode isconfigured]
      |  +--ro address           inet:ip-address
      |  +--ro local-mode        identityref
      |  +--ro isconfigured      boolean
      |  |  ...
      |  +--ro ntp-statistics
      |        ...
      |  +--rw interface* [name]
      |     +--rw name                if:interface-ref
      |     +--rw broadcast-server! {broadcast-server}?
      |     |  ...
      |     +--rw broadcast-client! {broadcast-client}?
      |     +--rw multicast-server* [address] {multicast-server}?
      |     |  +--rw address
      |     |  |       rt-types:ip-multicast-group-address
      |     |  |  ...
      |     +--rw multicast-client* [address] {multicast-client}?
      |     |  +--rw address    rt-types:ip-multicast-group-address
      |     +--rw manycast-server* [address] {manycast-server}?
      |     |  +--rw address    rt-types:ip-multicast-group-address
      |     +--rw manycast-client* [address] {manycast-client}?
      |        +--rw address
      |        |       rt-types:ip-multicast-group-address
      |        |  ...
      +--ro ntp-statistics
```

```
        +--...
```

The full data model tree for the YANG module "ietf-ntp" is in Appendix A.

This data model defines one top-level container which includes both the NTP configuration and the NTP running state including access rules, authentication, associations, unicast configurations, interfaces, system status and associations.

3. Relationship with NTPv4-MIB

If the device implements the NTPv4-MIB [RFC5907], data nodes 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 NTP Configuration Data Nodes and Related NTPv4-MIB Objects

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

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

YANG NTP State Data Nodes and Related NTPv4-MIB Objects

4. Relationship with RFC 7317

This section describes the relationship with NTP definition in Section 3.2 System Time Management of [RFC7317] . YANG data nodes in /ntp/ also support per-interface configuration which is not supported in /system/ntp. If the yang model defined in this document is implemented, then /system/ntp SHOULD NOT be used and MUST be ignored.

YANG data nodes in /ntp/	YANG data nodes in /system/ntp
ntp!	enabled
unicast-configuration	server
	server/name
unicast-configuration/address	server/transport/udp/address
unicast-configuration/port	server/transport/udp/port
unicast-configuration/type	server/association-type
unicast-configuration/iburst	server/iburst
unicast-configuration/prefer	server/prefer

YANG NTP Configuration Data Nodes and counterparts in RFC 7317 Objects

5. Access Rules

An Access Control List (ACL) is one of the basic elements used to configure device-forwarding behavior. An ACL is a user-ordered set of rules that is used to filter traffic on a networking device.

As per [RFC1305] and [RFC5905], NTP could include an access-control feature that prevents unauthorized access and controls which peers are allowed to update the local clock. Further it is useful to differentiate between the various kinds of access and attach a different acl-rule to each. For this, the YANG module allows such configuration via /ntp/access-rules. The access-rule itself is configured via [RFC8519].

Following access modes are supported -

- Peer: Permit others to synchronize their time with the NTP entity or it can synchornize its time with others. NTP control queries are also accepted.

- Server: Permit others to synchronize their time with the NTP entity, but vice versa is not supported. NTP control queries are accepted.

- Server-only: Permit others to synchronize their time with NTP entity, but vice versa is not supported. NTP control queries are not accepted.

- Query-only: Only control queries are accepted.

Query-only is the most restricted where as the peer is the full access authority. The ability to give different ACL rules for different access modes allows for a greater control by the operator.

6. Key Management

As per [RFC1305] and [RFC5905], when authentication is enabled, NTP employs a crypto-checksum, computed by the sender and checked by the receiver, together with a set of predistributed algorithms, and cryptographic keys indexed by a key identifier included in the NTP message. This key-id is a 32-bit unsigned integer that MUST be configured on the NTP peers before the authentication could be used. For this reason, this YANG module allows such configuration via /ntp/authentication/authentication-keys/. Further at the time of configuration of NTP association (for example unicast-server), the key-id is specified.

7. NTP Version

This YANG model allow a version to be configured for the NTP association i.e. an operator can control the use of NTPv3 [RFC1305] or NTPv4 [RFC5905] for each association it forms. This allows backward compatibility with a legacy system.

8. NTP YANG Module

```
<CODE BEGINS> file "ietf-ntp@2021-02-22.yang"
module ietf-ntp {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-ntp";
  prefix ntp;

  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types";
  }
  import ietf-inet-types {
```

```
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }
  import ietf-interfaces {
    prefix if;
    reference
      "RFC 8343: A YANG Data Model for Interface Management";
  }
  import ietf-system {
    prefix sys;
    reference
      "RFC 7317: A YANG Data Model for System Management";
  }
  import ietf-access-control-list {
    prefix acl;
    reference
      "RFC 8519: YANG Data Model for Network Access Control
       Lists (ACLs)";
  }
  import ietf-routing-types {
    prefix rt-types;
    reference
      "RFC 8294: Common YANG Data Types for the Routing Area";
  }
  import ietf-netconf-acm {
    prefix nacm;
    reference
      "RFC 8341: Network Configuration Protocol (NETCONF) Access
       Control Model";
  }

  organization
    "IETF NTP (Network Time Protocol) Working Group";
  contact
    "WG Web:  <http://tools.ietf.org/wg/ntp/>
     WG List:  <mailto: ntp@ietf.org
     Editor:   Dhruv Dhody
               <mailto:dhruv.ietf@gmail.com>
     Editor:   Ankit Kumar Sinha
               <mailto:ankit.ietf@gmail.com>";
  description
    "This document defines a YANG data model for Network Time Protocol
     (NTP) implementations. The data model includes configuration data
     and state data.

     Copyright (c) 2021 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
     (https://trustee.ietf.org/license-info).

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

     The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
     NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED',
     'MAY', and 'OPTIONAL' in this document are to be interpreted as
     described in BCP 14 (RFC 2119) (RFC 8174) when, and only when,
     they appear in all capitals, as shown here.";

  revision 2021-02-22 {
    description
      "Initial revision.";
    reference
      "RFC XXXX: A YANG Data Model for NTP.";
  }

  /* Note: The RFC Editor will replace XXXX with the number assigned
  to this document once it becomes an RFC.*/
  /* Typedef Definitions */

  typedef ntp-stratum {
    type uint8 {
      range "1..16";
    }
    description
      "The level of each server in the hierarchy is defined by
       a stratum. Primary servers are assigned with stratum
       one; secondary servers at each lower level are assigned with
       one stratum greater than the preceding level";
    reference
      "RFC 5905: Network Time Protocol Version 4: Protocol and
       Algorithms Specification, Section 3";
  }

  typedef ntp-version {
    type uint8;
    default "4";
    description
      "The current NTP version supported by corresponding
       association.";
    reference
```

```
      "RFC 5905: Network Time Protocol Version 4: Protocol and
       Algorithms Specification, Section 1";
  }

  typedef refid {
    type union {
      type inet:ipv4-address;
      type uint32;
      type string {
        length "4";
      }
    }
    description
      "A code identifying the particular server or reference
       clock.  The interpretation depends upon stratum. It
       could be an IPv4 address or first 32 bits of the MD5 hash of
       the IPv6 address or a string for the Reference Identifier
       and KISS codes. Some examples:
       -- a refclock ID like '127.127.1.0' for local clock sync
       -- uni/multi/broadcast associations for IPv4 will look like
       '203.0.113.1' and '0x4321FEDC' for IPv6
       -- sync with primary source will look like 'DCN', 'NIST',
       'ATOM'
       -- KISS codes will look like 'AUTH', 'DROP', 'RATE'
       Note that the use of MD5 hash for IPv6 address is not for
       cryptographic purposes ";
    reference
      "RFC 5905: Network Time Protocol Version 4: Protocol and
       Algorithms Specification, Section 7.3";
  }

  typedef ntp-date-and-time {
    type union {
      type yang:date-and-time;
      type uint8;
    }
    description
      "Follows the normal date-and-time format when valid value
       exist, otherwise allows for setting special value such as
       zero.";
  }

  /* features */

  feature ntp-port {
    description
      "Support for NTP port configuration";
    reference
```

```
      "RFC 5905: Network Time Protocol Version 4: Protocol and
       Algorithms Specification, Section 7.2";
  }

  feature authentication {
    description
      "Support for NTP symmetric key authentication";
    reference
      "RFC 5905: Network Time Protocol Version 4: Protocol and
       Algorithms Specification, Section 7.3";
  }

  feature deprecated {
    description
      "Support deprecated MD5-based authentication (RFC 8573) or
       SHA-1 or any other deprecated authentication.
       It is enabled to support legacy compatibility when secure
       cryptographic algorithm is not availaible to use.";
    reference
      "RFC 1321: The MD5 Message-Digest Algorithm
       RFC 3174: US Secure Hash Algorithm 1 (SHA1)";
  }

  feature hex-key-string {
    description
      "Support hexadecimal key string.";
  }

  feature access-rules {
    description
      "Support for NTP access control";
    reference
      "RFC 5905: Network Time Protocol Version 4: Protocol and
       Algorithms Specification, Section 9.2";
  }

  feature unicast-configuration {
    description
      "Support for NTP client/server or active/passive
       in unicast";
    reference
      "RFC 5905: Network Time Protocol Version 4: Protocol and
       Algorithms Specification, Section 3";
  }

  feature broadcast-server {
    description
      "Support for broadcast server";
```

```
    reference
      "RFC 5905: Network Time Protocol Version 4: Protocol and
       Algorithms Specification, Section 3";
  }

  feature broadcast-client {
    description
      "Support for broadcast client";
    reference
      "RFC 5905: Network Time Protocol Version 4: Protocol and
       Algorithms Specification, Section 3";
  }

  feature multicast-server {
    description
      "Support for multicast server";
    reference
      "RFC 5905: Network Time Protocol Version 4: Protocol and
       Algorithms Specification, Section 3.1";
  }

  feature multicast-client {
    description
      "Support for multicast client";
    reference
      "RFC 5905: Network Time Protocol Version 4: Protocol and
       Algorithms Specification, Section 3.1";
  }

  feature manycast-server {
    description
      "Support for manycast server";
    reference
      "RFC 5905: Network Time Protocol Version 4: Protocol and
       Algorithms Specification, Section 3.1";
  }

  feature manycast-client {
    description
      "Support for manycast client";
    reference
      "RFC 5905: Network Time Protocol Version 4: Protocol and
       Algorithms Specification, Section 3.1";
  }

  /* Identity */
  /* unicast-configurations types */
```

```
  identity unicast-configuration-type {
    if-feature "unicast-configuration";
    description
      "This defines NTP unicast mode of operation as used
       for unicast-configurations.";
  }

  identity uc-server {
    if-feature "unicast-configuration";
    base unicast-configuration-type;
    description
      "Use client association mode. This device
       will not provide synchronization to the
       configured NTP server.";
  }

  identity uc-peer {
    if-feature "unicast-configuration";
    base unicast-configuration-type;
    description
      "Use symmetric active association mode.
       This device may provide synchronization
       to the configured NTP server.";
  }

  /* association-modes */

  identity association-mode {
    description
      "The NTP association modes.";
    reference
      "RFC 5905: Network Time Protocol Version 4: Protocol and
       Algorithms Specification, Section 3";
  }

  identity active {
    base association-mode;
    description
      "Use symmetric active association mode (mode 1).
       This device may synchronize with its NTP peer,
       or provide synchronization to configured NTP peer.";
  }

  identity passive {
    base association-mode;
    description
      "Use symmetric passive association mode (mode 2).
       This device has learned this association dynamically.
```

```
       This device may synchronize with its NTP peer.";
  }

  identity client {
    base association-mode;
    description
      "Use client association mode (mode 3).
       This device will not provide synchronization
       to the configured NTP server.";
  }

  identity server {
    base association-mode;
    description
      "Use server association mode (mode 4).
       This device will provide synchronization to
       NTP clients.";
  }

  identity broadcast-server {
    base association-mode;
    description
      "Use broadcast server mode (mode 5).
       This mode defines that its either working
       as broadcast-server or multicast-server.";
  }

  identity broadcast-client {
    base association-mode;
    description
      "This mode defines that its either working
       as broadcast-client (mode 6) or multicast-client.";
  }

  /* access-mode */

  identity access-mode {
    if-feature "access-rules";
    description
      "This defines NTP access modes. These identifies
       how the ACL is applied with NTP.";
    reference
      "RFC 5905: Network Time Protocol Version 4: Protocol and
       Algorithms Specification, Section 9.2";
  }

  identity peer-access-mode {
    if-feature "access-rules";
```

```
    base access-mode;
    description
      "Permit others to synchronize their time with this NTP
       entity or it can synchornize its time with others.
       NTP control queries are also accepted. This enables
       full access authority.";
  }

  identity server-access-mode {
    if-feature "access-rules";
    base access-mode;
    description
      "Permit others to synchronize their time with this NTP
       entity, but vice versa is not supported. NTP control
       queries are accepted.";
  }

  identity server-only-access-mode {
    if-feature "access-rules";
    base access-mode;
    description
      "Permit others to synchronize their time with this NTP
       entity, but vice versa is not supported. NTP control
       queries are not accepted.";
  }

  identity query-only-access-mode {
    if-feature "access-rules";
    base access-mode;
    description
      "Only control queries are accepted.";
  }

  /* clock-state */

  identity clock-state {
    description
      "This defines NTP clock status at a high level.";
  }

  identity synchronized {
    base clock-state;
    description
      "Indicates that the local clock has been synchronized with
       an NTP server or the reference clock.";
  }

  identity unsynchronized {
```

```
    base clock-state;
    description
      "Indicates that the local clock has not been synchronized
       with any NTP server.";
  }

  /* ntp-sync-state */

  identity ntp-sync-state {
    description
      "This defines NTP clock sync state at a more granular
       level. Referred as 'Clock state definitions' in RFC 5905";
    reference
      "RFC 5905: Network Time Protocol Version 4: Protocol and
       Algorithms Specification, Appendix A.1.1";
  }

  identity clock-not-set {
    base ntp-sync-state;
    description
      "Indicates the clock is not updated.";
  }

  identity freq-set-by-cfg {
    base ntp-sync-state;
    description
      "Indicates the clock frequency is set by
       NTP configuration or file.";
  }

  identity spike {
    base ntp-sync-state;
    description
      "Indicates a spike is detected.";
  }

  identity freq {
    base ntp-sync-state;
    description
      "Indicates the frequency mode.";
  }

  identity clock-synchronized {
    base ntp-sync-state;
    description
      "Indicates that the clock is synchronized";
  }
```

```
  /* crypto-algorithm */

  identity crypto-algorithm {
    description
      "Base identity of cryptographic algorithm options.";
  }

  identity hmac-sha-1-12 {
    base crypto-algorithm;
    description
      "The HMAC-SHA1-12 algorithm.";
  }

  identity md5 {
    if-feature "deprecated";
    base crypto-algorithm;
    description
      "The MD5 algorithm. Note that RFC 8573
       deprecates the use of MD5-based authentication.";
  }

  identity sha-1 {
    if-feature "deprecated";
    base crypto-algorithm;
    description
      "The SHA-1 algorithm.";
  }

  identity hmac-sha-1 {
    base crypto-algorithm;
    description
      "HMAC-SHA-1 authentication algorithm.";
  }

  identity hmac-sha-256 {
    description
      "HMAC-SHA-256 authentication algorithm.";
  }

  identity hmac-sha-384 {
    description
      "HMAC-SHA-384 authentication algorithm.";
  }

  identity hmac-sha-512 {
    description
      "HMAC-SHA-512 authentication algorithm.";
  }
```

```
  identity aes-cmac {
    base crypto-algorithm;
    description
      "The AES-CMAC algorithm - required by
       RFC 8573 for MAC for the NTP";
    reference
      "RFC 4493: The AES-CMAC Algorithm";
  }

  /* Groupings */

  grouping key {
    description
      "The key.";
    nacm:default-deny-all;
    choice key-string-style {
      description
        "Key string styles";
      case keystring {
        leaf keystring {
          type string;
          description
            "Key string in ASCII format.";
        }
      }
      case hexadecimal {
        if-feature "hex-key-string";
        leaf hexadecimal-string {
          type yang:hex-string;
          description
            "Key in hexadecimal string format.  When compared
             to ASCII, specification in hexadecimal affords
             greater key entropy with the same number of
             internal key-string octets.  Additionally, it
             discourages usage of well-known words or
             numbers.";
        }
      }
    }
  }

  grouping authentication-key {
    description
      "To define an authentication key for a Network Time
       Protocol (NTP) time source.";
    nacm:default-deny-all;
    leaf key-id {
      type uint32 {
```

```
        range "1..max";
      }
      description
        "Authentication key identifier.";
    }
    leaf algorithm {
      type identityref {
        base crypto-algorithm;
      }
      description
        "Authentication algorithm. Note that RFC 8573
         deprecates the use of MD5-based authentication
         and recommends AES-CMAC.";
    }
    container key {
      uses key;
      description
        "The key. Note that RFC 8573 deprecates the use
         of MD5-based authentication.";
    }
    leaf istrusted {
      type boolean;
      description
        "Key-id is trusted or not";
    }
    reference
      "RFC 5905: Network Time Protocol Version 4: Protocol and
       Algorithms Specification, Section 7.3";
  }

  grouping authentication {
    description
      "Authentication.";
    choice authentication-type {
      description
        "Type of authentication.";
      case symmetric-key {
        leaf key-id {
          type leafref {
            path "/ntp:ntp/ntp:authentication/"
               + "ntp:authentication-keys/ntp:key-id";
          }
          description
            "Authentication key id referenced in this
             association.";
        }
      }
    }
```

```
  }

  grouping statistics {
    description
      "NTP packet statistic.";
    leaf discontinuity-time {
      type ntp-date-and-time;
      description
        "The time on the most recent occasion at which any one or
         more of this NTP counters suffered a discontinuity. If
         no such discontinuities have occurred, then this node
         contains the time the NTP association was
         (re-)initialized.";
    }
    leaf packet-sent {
      type yang:counter32;
      description
        "The total number of NTP packets delivered to the
         transport service by this NTP entity for this
         association.
         Discontinuities in the value of this counter can occur
         upon cold start or reinitialization of the NTP entity, the
         management system and at other times.";
    }
    leaf packet-sent-fail {
      type yang:counter32;
      description
        "The number of times NTP packets sending failed.";
    }
    leaf packet-received {
      type yang:counter32;
      description
        "The total number of NTP packets delivered to the
         NTP entity from this association.
         Discontinuities in the value of this counter can occur
         upon cold start or reinitialization of the NTP entity, the
         management system and at other times.";
    }
    leaf packet-dropped {
      type yang:counter32;
      description
        "The total number of NTP packets that were delivered
         to this NTP entity from this association and this entity
         was not able to process due to an NTP protocol error.
         Discontinuities in the value of this counter can occur
         upon cold start or reinitialization of the NTP entity, the
         management system and at other times.";
    }
```

```
  }

  grouping common-attributes {
    description
      "NTP common attributes for configuration.";
    leaf minpoll {
      type int8;
      default "6";
      description
        "The minimum poll interval used in this association.";
      reference
        "RFC 5905: Network Time Protocol Version 4: Protocol and
         Algorithms Specification, Section 7.2";
    }
    leaf maxpoll {
      type int8;
      default "10";
      description
        "The maximum poll interval used in this association.";
      reference
        "RFC 5905: Network Time Protocol Version 4: Protocol and
         Algorithms Specification, Section 7.2";
    }
    leaf port {
      if-feature "ntp-port";
      type inet:port-number {
        range "123 | 1025..max";
      }
      default "123";
      description
        "Specify the port used to send NTP packets.";
      reference
        "RFC 5905: Network Time Protocol Version 4: Protocol and
         Algorithms Specification, Section 7.2";
    }
    leaf version {
      type ntp-version;
      description
        "NTP version.";
    }
    reference
      "RFC 5905: Network Time Protocol Version 4: Protocol and
       Algorithms Specification";
  }

  grouping association-ref {
    description
      "Reference to NTP association mode";
```

```
    leaf associations-address {
      type leafref {
        path "/ntp:ntp/ntp:associations/ntp:address";
      }
      description
        "Indicates the association's address
         which result in clock synchronization.";
    }
    leaf associations-local-mode {
      type leafref {
        path "/ntp:ntp/ntp:associations/ntp:local-mode";
      }
      description
        "Indicates the association's local-mode
         which result in clock synchronization.";
    }
    leaf associations-isconfigured {
      type leafref {
        path "/ntp:ntp/ntp:associations/"
           + "ntp:isconfigured";
      }
      description
        "The association was configured or dynamic
         which result in clock synchronization.";
    }
  }

  /* Configuration data nodes */

  container ntp {
    when 'false() = boolean(/sys:system/sys:ntp)' {
      description
        "Applicable when the system /sys/ntp/ is not used.";
    }
    presence "NTP is enabled and system should attempt to
              synchronize the system clock with an NTP server
              from the 'ntp/associations' list.";
    description
      "Configuration parameters for NTP.";
    leaf port {
      if-feature "ntp-port";
      type inet:port-number {
        range "123 | 1025..max";
      }
      default "123";
      description
        "Specify the port used to send and receive NTP packets.";
      reference
```

```
        "RFC 5905: Network Time Protocol Version 4: Protocol and
         Algorithms Specification, Section 7.2";
    }
    container refclock-master {
      presence "NTP master clock is enabled.";
      description
        "Configures the local clock of this device as NTP server.";
      leaf master-stratum {
        type ntp-stratum;
        default "16";
        description
          "Stratum level from which NTP clients get their time
           synchronized.";
      }
    }
    container authentication {
      if-feature "authentication";
      description
        "Configuration of authentication.";
      leaf auth-enabled {
        type boolean;
        default "false";
        description
          "Controls whether NTP authentication is enabled
           or disabled on this device.";
      }
      list authentication-keys {
        key "key-id";
        uses authentication-key;
        description
          "List of authentication keys.";
      }
    }
    container access-rules {
      if-feature "access-rules";
      description
        "Configuration to control access to NTP service
         by using NTP access-group feature.
         The access-mode identifies how the acl is
         applied with NTP.";
      list access-rule {
        key "access-mode";
        description
          "List of access rules.";
        leaf access-mode {
          type identityref {
            base access-mode;
          }
```

```
          description
            "NTP access mode. The definition of each possible value:
             peer: Both time request and control query can be
             performed.
             server: Enables the server access and query.
             synchronization: Enables the server access only.
             query: Enables control query only.";
        }
        leaf acl {
          type leafref {
            path "/acl:acls/acl:acl/acl:name";
          }
          description
            "Control access configuration to be used.";
        }
        reference
          "RFC 5905: Network Time Protocol Version 4: Protocol and
           Algorithms Specification, Section 9.2";
      }
    }
    container clock-state {
      config false;
      description
        "Clock operational state of the NTP.";
      container system-status {
        description
          "System status of NTP.";
        leaf clock-state {
          type identityref {
            base clock-state;
          }
          mandatory true;
          description
            "The state of system clock. The definition of each
             possible value is:
             synchronized: Indicates local clock is synchronized.
             unsynchronized: Indicates local clock is not
             synchronized.";
        }
        leaf clock-stratum {
          type ntp-stratum;
          mandatory true;
          description
            "The NTP entity's own stratum value. Should be one greater
             than preceeding level. 16 if unsyncronized.";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification, Section 3";
```

```
        }
        leaf clock-refid {
          type refid;
          mandatory true;
          description
            "A code identifying the particular server or reference
             clock.  The interpretation depends upon stratum. It
             could be an IPv4 address or first 32 bits of the MD5 hash
             of the IPv6 address or a string for the Reference
             Identifier and KISS codes. Some examples:
             -- a refclock ID like '127.127.1.0' for local clock sync
             -- uni/multi/broadcast associations for IPv4 will look like
             '203.0.113.1' and '0x4321FEDC' for IPv6
             -- sync with primary source will look like 'DCN', 'NIST',
             'ATOM'
             -- KISS codes will look like 'AUTH', 'DROP', 'RATE'
             Note that the use of MD5 hash for IPv6 address is not for
             cryptographic purposes ";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification, Section 7.3";
        }
        uses association-ref {
          description
            "Reference to Association.";
        }
        leaf nominal-freq {
          type decimal64 {
            fraction-digits 4;
          }
          units "Hz";
          mandatory true;
          description
            "The nominal frequency of the local clock. An ideal
             frequency with zero uncertainty.";
        }
        leaf actual-freq {
          type decimal64 {
            fraction-digits 4;
          }
          units "Hz";
          mandatory true;
          description
            "The actual frequency of the local clock.";
        }
        leaf clock-precision {
          type int8;
          units "Hz";
```

```
          mandatory true;
          description
            "Clock precision of this system in integer format
             (prec=2^(-n)). A value of 5 would mean 2^-5 = 31.25 ms.";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification, Section 7.3";
        }
        leaf clock-offset {
          type decimal64 {
            fraction-digits 3;
          }
          units "milliseconds";
          description
            "The time offset to the current selected reference time
             source e.g., '0.032ms' or '1.232ms'.";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification, Section 9.1";
        }
        leaf root-delay {
          type decimal64 {
            fraction-digits 3;
          }
          units "milliseconds";
          description
            "Total delay along the path to root clock.";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification, Section 4 and 7.3";
        }
        leaf root-dispersion {
          type decimal64 {
            fraction-digits 3;
          }
          units "milliseconds";
          description
            "The dispersion between the local clock
             and the root clock, e.g., '6.927ms'.";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification, Section 4 and 7.3";
        }
        leaf reference-time {
          type ntp-date-and-time;
          description
            "The reference timestamp. Time when the system clock was
             last set or corrected";
```

```
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification, Section 7.3";
        }
        leaf sync-state {
          type identityref {
            base ntp-sync-state;
          }
          mandatory true;
          description
            "The synchronization status of the local clock. Referred to
             as 'Clock state definitions' in RFC 5905";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification, Appendix A.1.1";
        }
      }
    }
    list unicast-configuration {
      if-feature "unicast-configuration";
      key "address type";
      description
        "List of NTP unicast-configurations.";
      leaf address {
        type inet:ip-address;
        description
          "Address of this association.";
      }
      leaf type {
        type identityref {
          base unicast-configuration-type;
        }
        description
          "Use client association mode.  This device
           will not provide synchronization to the
           configured NTP server.";
      }
      container authentication {
        if-feature "authentication";
        description
          "Authentication used for this association.";
        uses authentication;
      }
      leaf prefer {
        type boolean;
        default "false";
        description
          "Whether this association is preferred or not.";
```

```
      }
      leaf burst {
        type boolean;
        default "false";
        description
          "If set, a series of packets are sent instead of a single
           packet within each synchronization interval to achieve
           faster synchronization.";
        reference
          "RFC 5905: Network Time Protocol Version 4: Protocol and
           Algorithms Specification, Section 13.1";
      }
      leaf iburst {
        type boolean;
        default "false";
        description
          "If set, a series of packets are sent instead of a single
           packet within the initial synchronization interval to
           achieve faster initial synchronization.";
        reference
          "RFC 5905: Network Time Protocol Version 4: Protocol and
           Algorithms Specification, Section 13.1";
      }
      leaf source {
        type if:interface-ref;
        description
          "The interface whose IP address is used by this association
           as the source address.";
      }
      uses common-attributes {
        description
          "Common attributes like port, version, min and max
           poll.";
      }
    }
    list associations {
      key "address local-mode isconfigured";
      config false;
      description
        "List of NTP associations. Here address, local-mode
         and isconfigured are required to uniquely identify
         a particular association. Lets take following examples -

         1) If RT1 acting as broadcast server,
         and RT2 acting as broadcast client, then RT2
         will form dynamic association with address as RT1,
         local-mode as client and isconfigured as false.
```

```
          2) When RT2 is configured
          with unicast-server RT1, then RT2 will form
          association with address as RT1, local-mode as client
          and isconfigured as true.

          Thus all 3 leaves are needed as key to unique identify
          the association.";
        leaf address {
          type inet:ip-address;
          description
            "The address of this association. Represents the IP
             address of a unicast/multicast/broadcast address.";
        }
        leaf local-mode {
          type identityref {
            base association-mode;
          }
          description
            "Local mode of this NTP association.";
        }
        leaf isconfigured {
          type boolean;
          description
            "Indicates if this association is configured or
             dynamically learned.";
        }
        leaf stratum {
          type ntp-stratum;
          description
            "The association stratum value.";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification, Section 3";
        }
        leaf refid {
          type refid;
          description
            "A code identifying the particular server or reference
             clock.  The interpretation depends upon stratum. It
             could be an IPv4 address or first 32 bits of the MD5 hash of
             the IPv6 address or a string for the Reference Identifier
             and KISS codes. Some examples:
             -- a refclock ID like '127.127.1.0' for local clock sync
             -- uni/multi/broadcast associations for IPv4 will look like
             '203.0.113.1' and '0x4321FEDC' for IPv6
             -- sync with primary source will look like 'DCN', 'NIST',
             'ATOM'
             -- KISS codes will look like 'AUTH', 'DROP', 'RATE'
```

```
          Note that the use of MD5 hash for IPv6 address is not for
          cryptographic purposes";
        reference
          "RFC 5905: Network Time Protocol Version 4: Protocol and
          Algorithms Specification, Section 7.3";
      }
      leaf authentication {
        if-feature "authentication";
        type leafref {
          path "/ntp:ntp/ntp:authentication/"
             + "ntp:authentication-keys/ntp:key-id";
        }
        description
          "Authentication Key used for this association.";
      }
      leaf prefer {
        type boolean;
        default "false";
        description
          "Indicates if this association is preferred.";
      }
      leaf peer-interface {
        type if:interface-ref;
        description
          "The interface which is used for communication.";
      }
      uses common-attributes {
        description
          "Common attributes like port, version, min and
          max poll.";
      }
      leaf reach {
        type uint8;
        description
          "The reachability of the configured
          server or peer.";
        reference
          "RFC 5905: Network Time Protocol Version 4: Protocol and
          Algorithms Specification, Section 9.2 and 13";
      }
      leaf unreach {
        type uint8;
        description
          "The unreachability of the configured
          server or peer.";
        reference
          "RFC 5905: Network Time Protocol Version 4: Protocol and
          Algorithms Specification, Section 9.2 and 13";
```

```
      }
      leaf poll {
        type int8;
        units "seconds";
        description
          "The polling interval for current association";
        reference
          "RFC 5905: Network Time Protocol Version 4: Protocol and
           Algorithms Specification, Section 7.3";
      }
      leaf now {
        type uint32;
        units "seconds";
        description
          "The time since the last NTP packet was
           received or last synchronized.";
      }
      leaf offset {
        type decimal64 {
          fraction-digits 3;
        }
        units "milliseconds";
        description
          "The offset between the local clock
           and the peer clock, e.g., '0.032ms' or '1.232ms'";
        reference
          "RFC 5905: Network Time Protocol Version 4: Protocol and
           Algorithms Specification, Section 8";
      }
      leaf delay {
        type decimal64 {
          fraction-digits 3;
        }
        units "milliseconds";
        description
          "The network delay between the local clock
           and the peer clock.";
        reference
          "RFC 5905: Network Time Protocol Version 4: Protocol and
           Algorithms Specification, Section 8";
      }
      leaf dispersion {
        type decimal64 {
          fraction-digits 3;
        }
        units "milliseconds";
        description
          "The root dispersion between the local clock
```

```
           and the peer clock.";
        reference
          "RFC 5905: Network Time Protocol Version 4: Protocol and
           Algorithms Specification, Section 10";
      }
      leaf originate-time {
        type ntp-date-and-time;
        description
          "This is the local time, in timestamp format,
           when latest NTP packet was sent to peer (called T1).";
        reference
          "RFC 5905: Network Time Protocol Version 4: Protocol and
           Algorithms Specification, Section 8";
      }
      leaf receive-time {
        type ntp-date-and-time;
        description
          "This is the local time, in timestamp format,
           when latest NTP packet arrived at peer (called T2).
           If the peer becomes unreachable the value is set to zero.";
        reference
          "RFC 5905: Network Time Protocol Version 4: Protocol and
           Algorithms Specification, Section 8";
      }
      leaf transmit-time {
        type ntp-date-and-time;
        description
          "This is the local time, in timestamp format,
           at which the NTP packet departed the peer (called T3).
           If the peer becomes unreachable the value is set to zero.";
        reference
          "RFC 5905: Network Time Protocol Version 4: Protocol and
           Algorithms Specification, Section 8";
      }
      leaf input-time {
        type ntp-date-and-time;
        description
          "This is the local time, in timestamp format,
           when the latest NTP message from the peer arrived (called
           T4). If the peer becomes unreachable the value is set to
           zero.";
        reference
          "RFC 5905: Network Time Protocol Version 4: Protocol and
           Algorithms Specification, Section 8";
      }
      container ntp-statistics {
        description
          "Per Peer packet send and receive statistics.";
```

```
        uses statistics {
          description
            "NTP send and receive packet statistics.";
        }
      }
    }
    container interfaces {
      description
        "Configuration parameters for NTP interfaces.";
      list interface {
        key "name";
        description
          "List of interfaces.";
        leaf name {
          type if:interface-ref;
          description
            "The interface name.";
        }
        container broadcast-server {
          if-feature "broadcast-server";
          presence "NTP broadcast-server is configured";
          description
            "Configuration of broadcast server.";
          leaf ttl {
            type uint8;
            description
              "Specifies the time to live (TTL) for a
               broadcast packet.";
            reference
              "RFC 5905: Network Time Protocol Version 4: Protocol and
               Algorithms Specification, Section 3.1";
          }
          container authentication {
            if-feature "authentication";
            description
              "Authentication used for this association.";
            uses authentication;
          }
          uses common-attributes {
            description
              "Common attributes such as port, version, min and
               max poll.";
          }
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification, Section 3.1";
        }
        container broadcast-client {
```

```
          if-feature "broadcast-client";
          presence "NTP broadcast-client is configured.";
          description
            "Configuration of broadcast-client.";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification, Section 3.1";
        }
        list multicast-server {
          if-feature "multicast-server";
          key "address";
          description
            "Configuration of multicast server.";
          leaf address {
            type rt-types:ip-multicast-group-address;
            description
              "The IP address to send NTP multicast packets.";
          }
          leaf ttl {
            type uint8;
            description
              "Specifies the time to live (TTL) for a
               multicast packet.";
            reference
              "RFC 5905: Network Time Protocol Version 4: Protocol and
               Algorithms Specification, Section 3.1";
          }
          container authentication {
            if-feature "authentication";
            description
              "Authentication used for this association.";
            uses authentication;
          }
          uses common-attributes {
            description
              "Common attributes such as port, version, min and
               max poll.";
          }
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification, Section 3.1";
        }
        list multicast-client {
          if-feature "multicast-client";
          key "address";
          description
            "Configuration of multicast-client.";
          leaf address {
```

```
            type rt-types:ip-multicast-group-address;
            description
              "The IP address of the multicast group to
               join.";
          }
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification, Section 3.1";
        }
        list manycast-server {
          if-feature "manycast-server";
          key "address";
          description
            "Configuration of manycast server.";
          leaf address {
            type rt-types:ip-multicast-group-address;
            description
              "The multicast group IP address to receive
               manycast client messages.";
          }
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification, Section 3.1";
        }
        list manycast-client {
          if-feature "manycast-client";
          key "address";
          description
            "Configuration of manycast-client.";
          leaf address {
            type rt-types:ip-multicast-group-address;
            description
              "The group IP address that the manycast client
               broadcasts the request message to.";
          }
          container authentication {
            if-feature "authentication";
            description
              "Authentication used for this association.";
            uses authentication;
          }
          leaf ttl {
            type uint8;
            description
              "Specifies the maximum time to live (TTL) for
               the expanding ring search.";
            reference
              "RFC 5905: Network Time Protocol Version 4: Protocol and
```

```
            Algorithms Specification, Section 3.1";
        }
        leaf minclock {
          type uint8;
          description
            "The minimum manycast survivors in this
             association.";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification, Section 13.2";
        }
        leaf maxclock {
          type uint8;
          description
            "The maximum manycast candidates in this
             association.";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification, Section 13.2";
        }
        leaf beacon {
          type int8;
          units "seconds";
          description
            "The beacon is the upper limit of poll interval. When the
             ttl reaches its limit without finding the minimum number
             of manycast servers, the poll interval increases until
             reaching the beacon value, when it starts over from the
             beginning.";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification, Section 13.2";
        }
        uses common-attributes {
          description
            "Common attributes like port, version, min and
             max poll.";
        }
        reference
          "RFC 5905: Network Time Protocol Version 4: Protocol and
           Algorithms Specification, Section 3.1";
      }
    }
  }
  container ntp-statistics {
    config false;
    description
      "Total NTP packet statistics.";
```

```
      uses statistics {
        description
          "NTP send and receive packet statistics.";
      }
    }
  }
}
<CODE ENDS>
```

9. Usage Example

This section include examples for illustration purposes.

Note: '\' line wrapping per [RFC8792].

9.1. Unicast association

This example describes how to configure a preferred unicast server present at 192.0.2.1 running at port 1025 with authentication-key 10 and version 4 (default).

```
  <edit-config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <target>
      <running/>
    </target>
    <config>
      <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
        <unicast-configuration>
          <address>192.0.2.1</address>
          <type>uc-server</type>
          <prefer>true</prefer>
          <port>1025</port>
          <authentication>
            <symmetric-key>
              <key-id>10</key-id>
            </symmetric-key>
          </authentication>
        </unicast-configuration>
      </ntp>
    </config>
  </edit-config>
```

An example with IPv6 would used the an IPv6 address (say 2001:db8::1) in the "address" leaf with no change in any other data tree.

```
    <edit-config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <target>
        <running/>
      </target>
      <config>
        <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
          <unicast-configuration>
            <address>2001:db8::1</address>
            <type>uc-server</type>
            <prefer>true</prefer>
            <port>1025</port>
            <authentication>
              <symmetric-key>
                <key-id>10</key-id>
              </symmetric-key>
            </authentication>
          </unicast-configuration>
        </ntp>
      </config>
    </edit-config>
```

This example is for retrieving unicast configurations -

```
<get>
  <filter type="subtree">
  <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <unicast-configuration>
      </unicast-configuration>
  </ntp>
  </filter>
</get>

<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
    <unicast-configuration>
      <address>192.0.2.1</address>
      <type>uc-server</type>
        <authentication>
          <symmetric-key>
            <key-id>10</key-id>
          </symmetric-key>
        </authentication>
      <prefer>true</prefer>
      <burst>false</burst>
      <iburst>true</iburst>
      <source/>
      <minpoll>6</minpoll>
      <maxpoll>10</maxpoll>
```

```
        <port>1025</port>
        <stratum>9</stratum>
        <refid>203.0.113.1</refid>
        <reach>255</reach>
        <unreach>0</unreach>
        <poll>128</poll>
        <now>10</now>
        <offset>0.025</offset>
        <delay>0.5</delay>
        <dispersion>0.6</dispersion>
        <originate-time>10-10-2017 07:33:55.253 Z+05:30\
        </originate-time>
        <receive-time>10-10-2017 07:33:55.258 Z+05:30\
        </receive-time>
        <transmit-time>10-10-2017 07:33:55.300 Z+05:30\
        </transmit-time>
        <input-time>10-10-2017 07:33:55.305 Z+05:30\
        </input-time>
        <ntp-statistics>
          <packet-sent>20</packet-sent>
          <packet-sent-fail>0</packet-sent-fail>
          <packet-received>20</packet-received>
          <packet-dropped>0</packet-dropped>
        </ntp-statistics>
      </unicast-configuration>
    </ntp>
  </data>
```

9.2. Refclock master

This example describes how to configure reference clock with stratum 8 -

```
<edit-config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <target>
    <running/>
  </target>
  <config>
    <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <refclock-master>
        <master-stratum>8</master-stratum>
      </refclock-master>
    </ntp>
  </config>
</edit-config>
```

This example describes how to get reference clock configuration -

```
<get>
  <filter type="subtree">
    <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <refclock-master>
      </refclock-master>
    </ntp>
  </filter>
</get>

<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
    <refclock-master>
      <master-stratum>8</master-stratum>
    </refclock-master>
  </ntp>
</data>
```

9.3. Authentication configuration

This example describes how to enable authentication and configure trusted authentication key 10 with mode as AES-CMAC and an hexadecimal string key -

```
<edit-config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <target>
    <running/>
  </target>
  <config>
    <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <authentication>
        <auth-enabled>true</auth-enabled>
        <authentication-keys>
          <key-id>10</key-id>
          <algorithm>aes-cmac</algorithm>
          <key>
            <hexadecimal-string>
              bb1d6929e95937287fa37d129b756746
            </hexadecimal-string>
          </key>
          <istrusted>true</istrusted>
        </authentication-keys>
      </authentication>
    </ntp>
  </config>
</edit-config>
```

This example describes how to get authentication related configuration -

```
<get>
  <filter type="subtree">
    <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <authentication>
      </authentication>
    </ntp>
  </filter>
</get>

<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
    <authentication>
      <auth-enabled>false</auth-enabled>
      <trusted-keys/>
      <authentication-keys>
        <key-id>10</key-id>
        <algorithm>aes-cmac</algorithm>
          <key>
            <hexadecimal-string>
              bb1d6929e95937287fa37d129b756746
            </hexadecimal-string>
          </key>
        <istrusted>true</istrusted>
      </authentication-keys>
    </authentication>
  </ntp>
</data>
```

9.4. Access configuration

This example describes how to configure access mode "peer" associated with acl 2000 -

```
<edit-config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <target>
    <running/>
  </target>
  <config>
    <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <access-rules>
        <access-rule>
          <access-mode>peer-access-mode</access-mode>
          <acl>2000</acl>
        </access-rule>
      </access-rules>
    </ntp>
  </config>
</edit-config>
```

This example describes how to get access related configuration -

```
<get>
  <filter type="subtree">
    <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <access-rules>
      </access-rules>
    </ntp>
  </filter>
</get>

<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
    <access-rules>
      <access-rule>
        <access-mode>peer-access-mode</access-mode>
        <acl>2000</acl>
      </access-rule>
    </access-rules>
  </ntp>
</data>
```

9.5. Multicast configuration

This example describes how to configure multicast-server with address as "224.0.1.1", port as 1025, and version as 3 and authentication keyid as 10 -

```
<edit-config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <target>
    <running/>
  </target>
  <config>
    <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <interfaces>
        <interface>
          <name>Ethernet3/0/0</name>
          <multicast-server>
            <address>224.0.1.1</address>
            <authentication>
              <symmetric-key>
                <key-id>10</key-id>
              </symmetric-key>
            </authentication>
            <port>1025</port>
            <version>3</version>
          </multicast-server>
        </interface>
      </interfaces>
    </ntp>
  </config>
</edit-config>
```

This example describes how to get multicast-server related configuration -

```
<get>
  <filter type="subtree">
    <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <interfaces>
        <interface>
          <multicast-server>
          </multicast-server>
        </interface>
      </interfaces>
    </ntp>
  </filter>
</get>

<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
    <interfaces>
      <interface>
        <name>Ethernet3/0/0</name>
        <multicast-server>
          <address>224.0.1.1</address>
          <ttl>8</ttl>
          <authentication>
            <symmetric-key>
              <key-id>10</key-id>
            </symmetric-key>
          </authentication>
          <minpoll>6</minpoll>
          <maxpoll>10</maxpoll>
          <port>1025</port>
          <version>3</version>
        </multicast-server>
      </interface>
    </interfaces>
  </ntp>
</data>
```

This example describes how to configure multicast-client with address as "224.0.1.1" -

```
<edit-config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <target>
    <running/>
  </target>
  <config>
    <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <interfaces>
        <interface>
          <name>Ethernet3/0/0</name>
          <multicast-client>
            <address>224.0.1.1</address>
          </multicast-client>
        </interface>
      </interfaces>
    </ntp>
  </config>
</edit-config>
```

This example describes how to get multicast-client related configuration -

```
<get>
  <filter type="subtree">
    <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <interfaces>
        <interface>
          <multicast-client>
          </multicast-client>
        </interface>
      </interfaces>
    </ntp>
  </filter>
</get>

<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
    <interfaces>
      <interface>
        <name>Ethernet3/0/0</name>
        <multicast-client>
          <address>224.0.1.1</address>
        </multicast-client>
      </interface>
    </interfaces>
  </ntp>
</data>
```

9.6. Manycast configuration

This example describes how to configure manycast-client with address as "224.0.1.1", port as 1025 and authentication keyid as 10 -

```
<edit-config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <target>
    <running/>
  </target>
  <config>
    <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <interfaces>
        <interface>
          <name>Ethernet3/0/0</name>
          <manycast-client>
            <address>224.0.1.1</address>
            <authentication>
              <symmetric-key>
                <key-id>10</key-id>
              </symmetric-key>
            </authentication>
            <port>1025</port>
          </manycast-client>
        </interface>
      </interfaces>
    </ntp>
  </config>
</edit-config>
```

This example describes how to get manycast-client related configuration -

```
<get>
  <filter type="subtree">
    <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <interfaces>
        <interface>
          <manycast-client>
          </manycast-client>
        </interface>
      </interfaces>
    </ntp>
  </filter>
</get>

<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
    <interfaces>
      <interface>
        <name>Ethernet3/0/0</name>
        <manycast-client>
          <address>224.0.1.1</address>
          <authentication>
            <symmetric-key>
              <key-id>10</key-id>
            </symmetric-key>
          </authentication>
          <ttl>8</ttl>
          <minclock>3</minclock>
          <maxclock>10</maxclock>
          <beacon>6</beacon>
          <minpoll>6</minpoll>
          <maxpoll>10</maxpoll>
          <port>1025</port>
        </manycast-client>
      </interface>
    </interfaces>
  </ntp>
</data>
```

This example describes how to configure manycast-server with address as "224.0.1.1" -

```
<edit-config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <target>
    <running/>
  </target>
  <config>
    <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <interfaces>
        <interface>
          <name>Ethernet3/0/0</name>
          <manycast-server>
            <address>224.0.1.1</address>
          </manycast-server>
        </interface>
      </interfaces>
    </ntp>
  </config>
</edit-config>
```

This example describes how to get manycast-server related configuration -

```
<get>
  <filter type="subtree">
    <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <interfaces>
        <interface>
          <manycast-server>
          </manycast-server>
        </interface>
      </interfaces>
    </ntp>
  </filter>
</get>

<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
    <interfaces>
      <interface>
        <name>Ethernet3/0/0</name>
        <manycast-server>
          <address>224.0.1.1</address>
        </manycast-server>
      </interface>
    </interfaces>
  </ntp>
</data>
```

9.7. Clock state

This example describes how to get clock current state -

```
<get>
  <filter type="subtree">
    <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <clock-state>
      </clock-state>
    </ntp>
  </filter>
</get>

<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
    <clock-state>
      <system-status>
        <clock-state>synchronized</clock-state>
        <clock-stratum>7</clock-stratum>
        <clock-refid>192.0.2.1</clock-refid>
        <associations-address>192.0.2.1\
        </associations-address>
        <associations-local-mode>client\
        </associations-local-mode>
        <associations-isconfigured>yes\
        </associations-isconfigured>
        <nominal-freq>100.0</nominal-freq>
        <actual-freq>100.0</actual-freq>
        <clock-precision>18</clock-precision>
        <clock-offset>0.025</clock-offset>
        <root-delay>0.5</root-delay>
        <root-dispersion>0.8</root-dispersion>
        <reference-time>10-10-2017 07:33:55.258 Z+05:30\
        </reference-time>
        <sync-state>clock-synchronized</sync-state>
      </system-status>
    </clock-state>
  </ntp>
</data>
```

9.8. Get all association

This example describes how to get all association present in the system -

```
   <get>
     <filter type="subtree">
       <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
         <associations>
         </associations>
       </ntp>
     </filter>
   </get>

   <data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
       <associations>
         <address>192.0.2.1</address>
         <stratum>9</stratum>
         <refid>203.0.113.1</refid>
         <local-mode>client</local-mode>
         <isconfigured>true</isconfigured>
         <authentication-key>10</authentication-key>
         <prefer>true</prefer>
         <peer-interface>Ethernet3/0/0</peer-interface>
         <minpoll>6</minpoll>
         <maxpoll>10</maxpoll>
         <port>1025</port>
         <version>4</version>
         <reach>255</reach>
         <unreach>0</unreach>
         <poll>128</poll>
         <now>10</now>
         <offset>0.025</offset>
         <delay>0.5</delay>
         <dispersion>0.6</dispersion>
         <originate-time>10-10-2017 07:33:55.253 Z+05:30\
         </originate-time>
         <receive-time>10-10-2017 07:33:55.258 Z+05:30\
         </receive-time>
         <transmit-time>10-10-2017 07:33:55.300 Z+05:30\
         </transmit-time>
         <input-time>10-10-2017 07:33:55.305 Z+05:30\
         </input-time>
         <ntp-statistics>
           <packet-sent>20</packet-sent>
           <packet-sent-fail>0</packet-sent-fail>
           <packet-received>20</packet-received>
           <packet-dropped>0</packet-dropped>
         </ntp-statistics>
       </associations>
     </ntp>
   </data>
```

9.9. Global statistic

This example describes how to get clock current state -

```
<get>
  <filter type="subtree">
    <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <ntp-statistics>
      </ntp-statistics>
    </ntp>
  </filter>
</get>

<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
    <ntp-statistics>
      <packet-sent>30</packet-sent>
      <packet-sent-fail>5</packet-sent-fail>
      <packet-received>20</packet-received>
      <packet-dropped>2</packet-dropped>
    </ntp-statistics>
  </ntp>
</data>
```

10. 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 IESG.

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

Note: The RFC Editor will replace XXXX with the number assigned to this document once it becomes an RFC.

11. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The NETCONF Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

There are a number of data nodes defined in this YANG module that 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. These are the subtrees and data nodes and their sensitivity/vulnerability:

/ntp/port - This data node specify the port number to be used to send NTP packets. Unexpected changes could lead to disruption and/or network misbehavior.

/ntp/authentication and /ntp/access-rules - The entries in the list include the authentication and access control configurations. Care should be taken while setting these parameters.

/ntp/unicast-configuration - The entries in the list include all unicast configurations (server or peer mode), and indirectly creates or modify the NTP associations. Unexpected changes could lead to disruption and/or network misbehavior.

/ntp/interfaces/interface - The entries in the list include all per-interface configurations related to broadcast, multicast and manycast mode, and indirectly creates or modify the NTP associations. Unexpected changes could lead to disruption and/or network misbehavior.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or

notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

/ntp/authentication/authentication-keys - The entries in the list includes all the NTP authentication keys. This information is sensitive and can be exploited and thus unauthorized access to this needs to be curtailed.

/ntp/associations - The entries in the list includes all active NTP associations of all modes. Unauthorized access to this also needs to be curtailed.

The leaf /ntp/authentication/authentication-keys/algorithm can be set to cryptographic algorithms that are no longer considered to be secure. As per [RFC8573], AES-CMAC is the recommended algorithm.

12. Acknowledgments

The authors would like to express their thanks to Sladjana Zoric, Danny Mayer, Harlan Stenn, Ulrich Windl, Miroslav Lichvar, Maurice Angermann, Watson Ladd, and Rich Salz for their review and suggestions.

Thanks to Andy Bierman for the YANG doctor review.

Thanks to Dieter Sibold for being the document shepherd and Erik Kline for being the responsible AD.

Thanks to Takeshi Takahashi for SECDIR review. Thanks to Tim Evens for GENART review.

A special thanks to Tom Petch for a very detailed YANG review and providing great suggestions for improvements.

13. References

13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

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

[RFC5907] Gerstung, H., Elliott, C., and B. Haberman, Ed., "Definitions of Managed Objects for Network Time Protocol Version 4 (NTPv4)", RFC 5907, DOI 10.17487/RFC5907, June 2010, <https://www.rfc-editor.org/info/rfc5907>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/info/rfc6991>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

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

[RFC8177] Lindem, A., Ed., Qu, Y., Yeung, D., Chen, I., and J. Zhang, "YANG Data Model for Key Chains", RFC 8177, DOI 10.17487/RFC8177, June 2017, <https://www.rfc-editor.org/info/rfc8177>.

[RFC8294] Liu, X., Qu, Y., Lindem, A., Hopps, C., and L. Berger, "Common YANG Data Types for the Routing Area", RFC 8294, DOI 10.17487/RFC8294, December 2017, <https://www.rfc-editor.org/info/rfc8294>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <https://www.rfc-editor.org/info/rfc8343>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8519] Jethanandani, M., Agarwal, S., Huang, L., and D. Blair, "YANG Data Model for Network Access Control Lists (ACLs)", RFC 8519, DOI 10.17487/RFC8519, March 2019, <https://www.rfc-editor.org/info/rfc8519>.

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

13.2. Informative References

[RFC1305] Mills, D., "Network Time Protocol (Version 3) Specification, Implementation and Analysis", RFC 1305, DOI 10.17487/RFC1305, March 1992, <https://www.rfc-editor.org/info/rfc1305>.

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

[RFC3174] Eastlake 3rd, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, DOI 10.17487/RFC3174, September 2001, <https://www.rfc-editor.org/info/rfc3174>.

[RFC4493] Song, JH., Poovendran, R., Lee, J., and T. Iwata, "The AES-CMAC Algorithm", RFC 4493, DOI 10.17487/RFC4493, June 2006, <https://www.rfc-editor.org/info/rfc4493>.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <https://www.rfc-editor.org/info/rfc7317>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

[RFC8792] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <https://www.rfc-editor.org/info/rfc8792>.

Appendix A. Full YANG Tree

The full tree for ietf-ntp YANG model is -

```
module: ietf-ntp
  +--rw ntp!
     +--rw port?                    inet:port-number {ntp-port}?
     +--rw refclock-master!
     |  +--rw master-stratum?   ntp-stratum
     +--rw authentication {authentication}?
     |  +--rw auth-enabled?          boolean
     |  +--rw authentication-keys* [key-id]
     |     +--rw key-id       uint32
     |     +--rw algorithm?   identityref
     |     +--rw key
     |     |  +--rw (key-string-style)?
     |     |     +--:(keystring)
     |     |     |  +--rw keystring?            string
     |     |     +--:(hexadecimal) {hex-key-string}?
     |     |        +--rw hexadecimal-string?   yang:hex-string
     |     +--rw istrusted?   boolean
     +--rw access-rules {access-rules}?
     |  +--rw access-rule* [access-mode]
     |     +--rw access-mode    identityref
     |     +--rw acl?           -> /acl:acls/acl/name
     +--ro clock-state
     |  +--ro system-status
     |     +--ro clock-state                  identityref
     |     +--ro clock-stratum                ntp-stratum
     |     +--ro clock-refid                  refid
```

```
|      +--ro associations-address?
|      |       -> /ntp/associations/address
|      +--ro associations-local-mode?
|      |       -> /ntp/associations/local-mode
|      +--ro associations-isconfigured?
|      |       -> /ntp/associations/isconfigured
|      +--ro nominal-freq                 decimal64
|      +--ro actual-freq                  decimal64
|      +--ro clock-precision              int8
|      +--ro clock-offset?                decimal64
|      +--ro root-delay?                  decimal64
|      +--ro root-dispersion?             decimal64
|      +--ro reference-time?              ntp-date-and-time
|      +--ro sync-state                   identityref
|    +--rw unicast-configuration* [address type]
|    |       {unicast-configuration}?
|    |  +--rw address           inet:ip-address
|    |  +--rw type              identityref
|    |  +--rw authentication {authentication}?
|    |  |  +--rw (authentication-type)?
|    |  |     +--:(symmetric-key)
|    |  |        +--rw key-id?   leafref
|    |  +--rw prefer?           boolean
|    |  +--rw burst?            boolean
|    |  +--rw iburst?           boolean
|    |  +--rw source?           if:interface-ref
|    |  +--rw minpoll?          int8
|    |  +--rw maxpoll?          int8
|    |  +--rw port?             inet:port-number {ntp-port}?
|    |  +--rw version?          ntp-version
|    +--ro associations* [address local-mode isconfigured]
|    |  +--ro address           inet:ip-address
|    |  +--ro local-mode        identityref
|    |  +--ro isconfigured      boolean
|    |  +--ro stratum?          ntp-stratum
|    |  +--ro refid?            refid
|    |  +--ro authentication?
|    |  |       -> /ntp/authentication/authentication-keys/key-id
|    |  |       {authentication}?
|    |  +--ro prefer?           boolean
|    |  +--ro peer-interface?   if:interface-ref
|    |  +--ro minpoll?          int8
|    |  +--ro maxpoll?          int8
|    |  +--ro port?             inet:port-number {ntp-port}?
|    |  +--ro version?          ntp-version
|    |  +--ro reach?            uint8
|    |  +--ro unreach?          uint8
|    |  +--ro poll?             int8
```

```
|  +--ro now?              uint32
|  +--ro offset?           decimal64
|  +--ro delay?            decimal64
|  +--ro dispersion?       decimal64
|  +--ro originate-time?   ntp-date-and-time
|  +--ro receive-time?     ntp-date-and-time
|  +--ro transmit-time?    ntp-date-and-time
|  +--ro input-time?       ntp-date-and-time
|  +--ro ntp-statistics
|     +--ro discontinuity-time?   ntp-date-and-time
|     +--ro packet-sent?          yang:counter32
|     +--ro packet-sent-fail?     yang:counter32
|     +--ro packet-received?      yang:counter32
|     +--ro packet-dropped?       yang:counter32
+--rw interfaces
|  +--rw interface* [name]
|     +--rw name                if:interface-ref
|     +--rw broadcast-server! {broadcast-server}?
|     |  +--rw ttl?              uint8
|     |  +--rw authentication {authentication}?
|     |  |  +--rw (authentication-type)?
|     |  |     +--:(symmetric-key)
|     |  |        +--rw key-id?   leafref
|     |  +--rw minpoll?          int8
|     |  +--rw maxpoll?          int8
|     |  +--rw port?             inet:port-number {ntp-port}?
|     |  +--rw version?          ntp-version
|     +--rw broadcast-client! {broadcast-client}?
|     +--rw multicast-server* [address] {multicast-server}?
|     |  +--rw address
|     |  |       rt-types:ip-multicast-group-address
|     |  +--rw ttl?              uint8
|     |  +--rw authentication {authentication}?
|     |  |  +--rw (authentication-type)?
|     |  |     +--:(symmetric-key)
|     |  |        +--rw key-id?   leafref
|     |  +--rw minpoll?          int8
|     |  +--rw maxpoll?          int8
|     |  +--rw port?             inet:port-number {ntp-port}?
|     |  +--rw version?          ntp-version
|     +--rw multicast-client* [address] {multicast-client}?
|     |  +--rw address    rt-types:ip-multicast-group-address
|     +--rw manycast-server* [address] {manycast-server}?
|     |  +--rw address    rt-types:ip-multicast-group-address
|     +--rw manycast-client* [address] {manycast-client}?
|        +--rw address
|        |       rt-types:ip-multicast-group-address
|        +--rw authentication {authentication}?
```

```
         |         |  +--rw (authentication-type)?
         |         |     +--:(symmetric-key)
         |         |        +--rw key-id?   leafref
         |         +--rw ttl?              uint8
         |         +--rw minclock?         uint8
         |         +--rw maxclock?         uint8
         |         +--rw beacon?           int8
         |         +--rw minpoll?          int8
         |         +--rw maxpoll?          int8
         |         +--rw port?             inet:port-number {ntp-port}?
         |         +--rw version?          ntp-version
         +--ro ntp-statistics
            +--ro discontinuity-time?   ntp-date-and-time
            +--ro packet-sent?          yang:counter32
            +--ro packet-sent-fail?     yang:counter32
            +--ro packet-received?      yang:counter32
            +--ro packet-dropped?       yang:counter32
```

Authors' Addresses

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

Email: eric.wu@huawei.com

Dhruv Dhody (editor)
Huawei
Divyashree Techno Park, Whitefield
Bangalore, Kanataka 560066
India

Email: dhruv.ietf@gmail.com

Ankit kumar Sinha (editor)
RtBrick Inc.
Bangalore, Kanataka
India

Email: ankit.ietf@gmail.com

Anil Kumar S N
RtBrick Inc.
Bangalore, Kanataka
India

Email: anil.ietf@gmail.com

Yi Zhao
Ericsson
China Digital Kingdom Bld., No.1 WangJing North Rd.
Beijing 100102
China

Email: yi.z.zhao@ericsson.com

Internet Engineering Task Force
Internet-Draft
Updates: 5905 (if approved)
Intended status: Standards Track
Expires: March 20, 2021

M. Lichvar
Red Hat

Sep 16, 2020

Alternative NTP port
draft-mlichvar-ntp-alternative-port-02

Abstract

This document updates RFC 5905 to specify an alternative port for the Network Time Protocol (NTP) which is restricted to NTP messages that do not allow traffic amplification in order to make NTP safe for the Internet.

Status of This Memo

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

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

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

This Internet-Draft will expire on March 20, 2021.

Copyright Notice

Table of Contents

1. Introduction

There are several modes specified for NTP. NTP packets in versions 2, 3, and 4 have a 3-bit field for the mode. Modes 1 (active), 2 (passive), 3 (client), 4 (server), and 5 (broadcast) are used for synchronization of clocks. They are specified in RFC 5905 [RFC5905]. Modes 6 and 7 are used for other purposes, like monitoring and remote management of NTP servers and clients. The mode 6 is specified in Control Messages Protocol for Use with Network Time Protocol Version 4 [I-D.ietf-ntp-mode-6-cmds].

The first group of modes typically does not allow any traffic amplification, i.e. the response is not larger than the request. An exception is Autokey specified in RFC 5906 [RFC5906]. Autokey is rarely supported on public NTP servers.

However, the modes 6 and 7 allow significant traffic amplification, which has been exploited in large-scale denial-of-service (DoS) attacks over the Internet.

Over time, network operators have been observed to implement the following mitigations:

1. Blocked UDP packets with destination or source port 123

2. Blocked UDP packets with destination or source port 123 and specific length (e.g. longer than 48 octets)

3. Blocked UDP packets with destination or source port 123 and NTP mode 6 or 7

4. Limited rate of UDP packets with destination or source port 123

From those, only the 3rd approach does not have an impact on synchronization of clocks with NTP.

The number of public servers in the pool.ntp.org project has dropped in large part due to the mitigations (citation?).

Longer NTP packets (using extension fields) are needed by NTS [I-D.ietf-ntp-using-nts-for-ntp].

This document specifies an alternative port for NTP which is restricted to the safe modes in order to enable synchronization of clocks in networks where the port 123 is blocked or rate limited.

1.1. Requirements Language

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

2. Alternative port - update to RFC 5905

The table in "Figure 6: Global Parameters" in Section 7.2 of [RFC5905] is extended with:

Name	Value	Description
ALTPORT	TBD	Alternative NTP port

The following text from Section 9.1 of [RFC5905]:

> srcport: UDP port number of the server or reference clock. This becomes the destination port number in packets sent from this association. When operating in symmetric modes (1 and 2), this field must contain the NTP port number PORT (123) assigned by the IANA. In other modes, it can contain any number consistent with local policy.

is replaced with:

> srcport: UDP port number of the server or reference clock. This becomes the destination port number in packets sent from this association. When operating in symmetric modes (1 and 2), this field must contain the NTP port number PORT (123) or the alternative NTP port ALTPORT (TBD) assigned by the IANA. In other modes, it can contain any number consistent with local policy.

The following text is added to the Section 9.1:

The port ALTPORT (TBD) is an alternative port to the port PORT (123). The protocol and the format of NTP packets sent from and to this port is unchanged. Both NTP requests and responses MAY be sent from the alternative port. An NTP packet MUST NOT be sent from the alternative port if it is a response which has a longer UDP payload than the request, or the number of NTP packets in a single response is larger than one.

Only modes 1 (active), 2 (passive), 3 (client), 4 (server), and 5 (broadcast) are generally usable on this port.

An NTP server SHOULD receive requests in the client mode on both the PORT (123) and ALTPORT (TBD) ports. If it responds, it MUST send the response from the port which received the request. If the server supports any extension fields in NTP packets, it MUST verify that each response is not larger than the request, even if the number of extension fields is constant and they have a constant length.

When an NTP client is started, it SHOULD send the first request to the alternative port. The client SHOULD be switching between the two ports until a valid response is received. The client MAY send a limited number of requests to both ports at the same time in order to speed up the discovery of the responding port. When both ports are responding, the client SHOULD prefer the alternative port.

An NTP server which supports NTS SHOULD include the NTPv4 Port Negotiation record in NTS-KE responses to specify the alternative port as the port to which the client should send NTP requests.

In the symmetric modes (active and passive) NTP packets are considered to be requests and responses at the same time. Therefore, the peers MUST send packets with an equal length in order to synchronize with each other. The peers MAY use different polling intervals (packets sent at subsequent polls are considered to be separate requests and responses).

3. IANA Considerations

IANA is requested to allocate the following port in the Service Name and Transport Protocol Port Number Registry [RFC6335]:

Service Name: ntp-alt

Transport Protocol: udp

Assignee: IESG <iesg@ietf.org>

Contact: IETF Chair <chair@ietf.org>

Description: Network Time Protocol

Reference: [[this memo]]

Port Number: [[TBD]], selected by IANA from the System Port range

4. Security Considerations

A Man-in-the-middle (MITM) attacker can selectively block requests sent to the alternative port to force a client to select the original port and get a degraded NTP service with a significant packet loss. The client needs to periodically try the alternative port to recover from the degraded service when the attack stops.

5. Acknowledgements

The author would like to thank Daniel Franke, Dhruv Dhody, and Ragnar Sundblad for their useful comments.

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

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

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <https://www.rfc-editor.org/info/rfc6335>.

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

6.2. Informative References

[I-D.ietf-ntp-mode-6-cmds]
Haberman, B., "Control Messages Protocol for Use with Network Time Protocol Version 4", draft-ietf-ntp-mode-6-cmds-09 (work in progress), June 2020.

[I-D.ietf-ntp-using-nts-for-ntp]
Franke, D., Sibold, D., Teichel, K., Dansarie, M., and R. Sundblad, "Network Time Security for the Network Time Protocol", draft-ietf-ntp-using-nts-for-ntp-28 (work in progress), March 2020.

[RFC5906] Haberman, B., Ed. and D. Mills, "Network Time Protocol Version 4: Autokey Specification", RFC 5906, DOI 10.17487/RFC5906, June 2010, <https://www.rfc-editor.org/info/rfc5906>.

Author's Address

Miroslav Lichvar
Red Hat
Purkynova 115
Brno 612 00
Czech Republic

Email: mlichvar@redhat.com