Internet Engineering Task Force
Internet-Draft
Updates: 5905 (if approved)
Intended status: Standards Track
Expires: April 15, 2021

M. Lichvar
Red Hat

Oct 12, 2020

Alternative NTP port
draft-ietf-ntp-alternative-port-00

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 April 15, 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 [RFC5906], which allows an NTP response to be longer than the request, e.g. packets containing the Certificate Message or Cookie Message extension field. Autokey is rarely used. If it is enabled on a publicly accessible server, the access needs to be tightly controlled to limit denial-of-service (DoS) attacks exploiting the amplification.

The modes 6 and 7 of NTP allow significant traffic amplification, which has been exploited in large-scale DoS attacks on the Internet. Publicly accessible servers that support these modes need to be configured to not respond to requests using the modes, as recommended in BCP 233 [RFC8633], but the number of servers that still do that is significant enough to require specific mitigations.

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?).

The length-specific filtering and rate limiting has an impact on the Network Time Security NTS [RFC8915] authentication, which uses extension fields in NTPv4 packets.

This document specifies an alternative port for NTP which is restricted to a subset of the NTP protocol which does not allow amplification in order to enable safe 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 that supports the alternative port MUST 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 support an NTP extension field, it MUST verify for each response that it is not longer than the request.
>
> 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, two peers using the alternative port MUST send packets with an equal length in order to synchronize with each other. The peers MAY still use different polling intervals as 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-10 (work in progress), September 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>.

[RFC8633] Reilly, D., Stenn, H., and D. Sibold, "Network Time Protocol Best Current Practices", BCP 223, RFC 8633, DOI 10.17487/RFC8633, July 2019, <https://www.rfc-editor.org/info/rfc8633>.

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

Author's Address

Miroslav Lichvar
Red Hat
Purkynova 115
Brno 612 00
Czech Republic

Email: mlichvar@redhat.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: March 7, 2021

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

September 3, 2020

A Secure Selection and Filtering Mechanism for the Network Time Protocol Version 4
draft-ietf-ntp-chronos-01

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.

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 all NTP communications are encrypted and authenticated. This document introduces an improved system process that incorporates an algorithm 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 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 Choronos 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 100ms 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.

Notaion	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
tc	The current time [sec], as indicated by the virtual clock value that is computed by Chronos

Table 1: Chronos Notations

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.

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 third of the samples with the lowest offset value and a third of the samples with the highest offset value 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 Chronos 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 atack 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 clock as part of the watchdog process.

According to empirical observations (presented in [Chronos_paper]), 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.

4. Chronos' Pseudocode

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

```
counter := 0
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 updates the clock so 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. When under attack, Chronos, which changes the list of the sampled servers more frequently than NTPv4 [Chronos_paper], and without using some of the filters in NTPv4's system process, can potentially be less precise (though provably more accurate and 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 offset and the offset computed by Chronos, as explained above, and checking whether it exceeds a certain threshold (10ms 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., 100ms), even a powerful man-in-the-middle 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>.

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, DOI 10.17487/RFC2629, June 1999, <https://www.rfc-editor.org/info/rfc2629>.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <https://www.rfc-editor.org/info/rfc3552>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <https://www.rfc-editor.org/info/rfc5226>.

[roughtime]
Patton, C., "Roughtime: Securing Time with Digital Signatures", 2018, <https://blog.cloudflare.com/roughtime/>.

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).

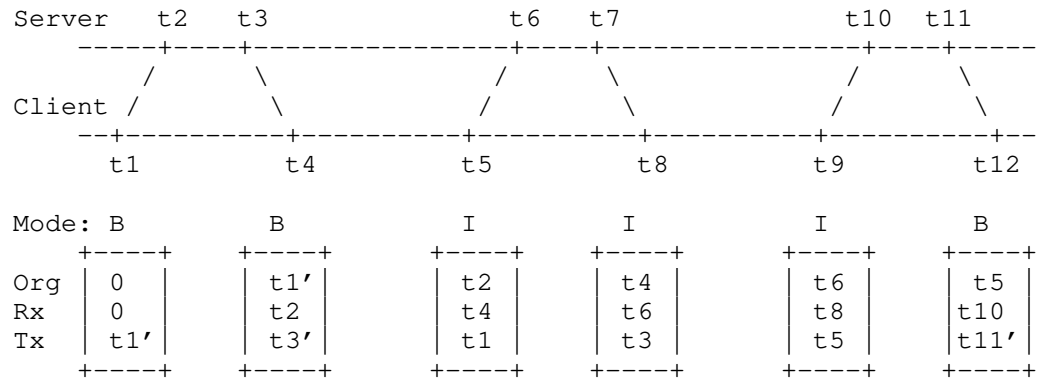

Figure 1: Packet timestamps in interleaved client/server mode

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.

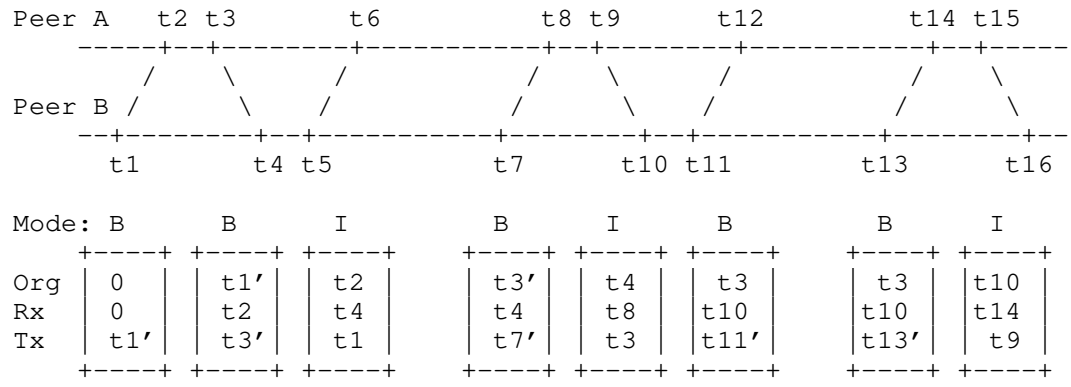

Figure 2: Packet timestamps in interleaved symmetric mode

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.

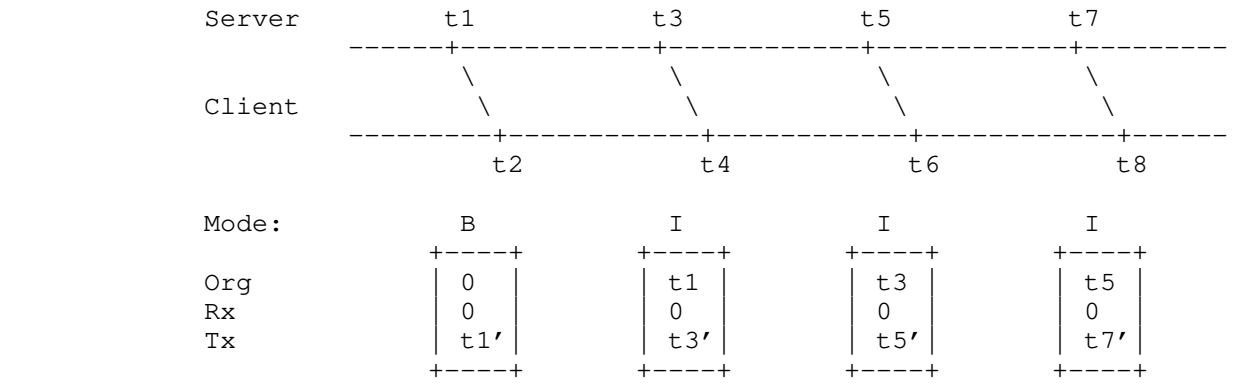

Figure 3: Packet timestamps in interleaved broadcast mode

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 Working Group
Internet-Draft
Intended status: Historic
Expires: April 1, 2021

B. Haberman, Ed.
JHU

September 28, 2020

Control Messages Protocol for Use with Network Time Protocol Version 4
draft-ietf-ntp-mode-6-cmds-10

Abstract

This document describes the structure of the control messages that were historically used with the Network Time Protocol before the advent of more modern control and management approaches. These control messages have been used to monitor and control the Network Time Protocol application running on any IP network attached computer. The information in this document was originally described in Appendix B of RFC 1305. The goal of this document is to provide an updated description of the control messages described in RFC 1305 in order to conform with the updated Network Time Protocol specification documented in RFC 5905.

The publication of this document is not meant to encourage the development and deployment of these control messages. This document is only providing a current reference for these control messages given the current status of RFC 1305.

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 April 1, 2021.

Table of Contents

1. Introduction

RFC 1305 [RFC1305] described a set of control messages for use within the Network Time Protocol (NTP) when a comprehensive network management solution was not available. The definitions of these control messages were not promulgated to RFC 5905 [RFC5905] when NTP version 4 was documented. These messages were intended for use only in systems where no other management facilities were available or appropriate, such as in dedicated-function bus peripherals. Support for these messages is not required in order to conform to RFC 5905 [RFC5905]. The control messages are described here as a current reference for use with an RFC 5905 implementation of NTP.

The publication of this document is not meant to encourage the development and deployment of these control messages. This document is only providing a current reference for these control messages given the current status of RFC 1305.

1.1. Control Message Overview

The NTP Mode 6 control messages are used by NTP management programs (e.g., ntpq) when a more robust network management facility (e.g., SNMP) is not available. These control messages provide rudimentary control and monitoring functions to manage a running instance of an NTP server. These commands are not designed to be used for communication between instances of running NTP servers.

The NTP Control Message has the value 6 specified in the mode field of the first octet of the NTP header and is formatted as shown in Figure 1. The format of the data field is specific to each command or response; however, in most cases the format is designed to be constructed and viewed by humans and so is coded in free-form ASCII. This facilitates the specification and implementation of simple management tools in the absence of fully evolved network-management facilities. As in ordinary NTP messages, the authenticator field follows the data field. If the authenticator is used the data field is zero-padded to a 32-bit boundary, but the padding bits are not considered part of the data field and are not included in the field count.

IP hosts are not required to reassemble datagrams over a certain size (576 octets for IPv4 [RFC0791] and 1280 octets for IPv6 [RFC2460]); however, some commands or responses may involve more data than will fit into a single datagram. Accordingly, a simple reassembly feature is included in which each octet of the message data is numbered starting with zero. As each fragment is transmitted the number of its first octet is inserted in the offset field and the number of

octets is inserted in the count field. The more-data (M) bit is set in all fragments except the last.

Most control functions involve sending a command and receiving a response, perhaps involving several fragments. The sender chooses a distinct, nonzero sequence number and sets the status field and "R" and "E" bits to zero. The responder interprets the opcode and additional information in the data field, updates the status field, sets the "R" bit to one and returns the three 32-bit words of the header along with additional information in the data field. In case of invalid message format or contents the responder inserts a code in the status field, sets the "R" and "E" bits to one and, optionally, inserts a diagnostic message in the data field.

Some commands read or write system variables (e.g., s.offset) and peer variables (e.g., p.stratum) for an association identified in the command. Others read or write variables associated with a radio clock or other device directly connected to a source of primary synchronization information. To identify which type of variable and association the Association ID is used. System variables are indicated by the identifier zero. As each association is mobilized a unique, nonzero identifier is created for it. These identifiers are used in a cyclic fashion, so that the chance of using an old identifier which matches a newly created association is remote. A management entity can request a list of current identifiers and subsequently use them to read and write variables for each association. An attempt to use an expired identifier results in an exception response, following which the list can be requested again.

Some exception events, such as when a peer becomes reachable or unreachable, occur spontaneously and are not necessarily associated with a command. An implementation may elect to save the event information for later retrieval or to send an asynchronous response (called a trap) or both. In case of a trap the IP address and port number is determined by a previous command and the sequence field is set as described below. Current status and summary information for the latest exception event is returned in all normal responses. Bits in the status field indicate whether an exception has occurred since the last response and whether more than one exception has occurred.

Commands need not necessarily be sent by an NTP peer, so ordinary access-control procedures may not apply; however, the optional mask/match mechanism suggested in Section Section 6 elsewhere in this document provides the capability to control access by mode number, so this could be used to limit access for control messages (mode 6) to selected address ranges.

1.2. Remote Facility Message Overview

The original development of the NTP daemon included a remote facility for monitoring and configuration. This facility used mode 7 commands to communicate with the NTP daemon. This document illustrates the mode 7 packet format only. The commands embedded in the mode 7 messages are implementation specific and not standardized in any way. The mode 7 message format is described in Appendix A.

2. NTP Control Message Format

The format of the NTP Control Message header, which immediately follows the UDP header, is shown in Figure 1. Following is a description of its fields.

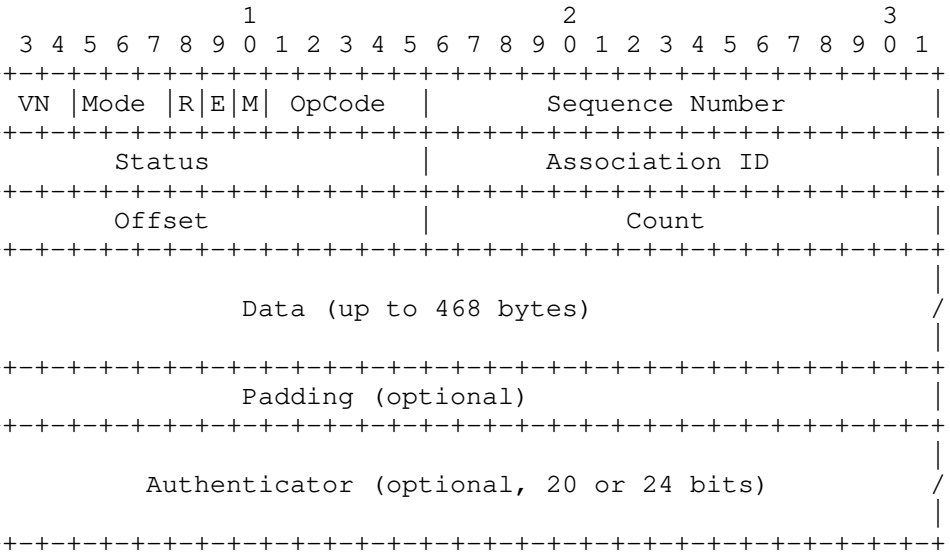

Figure 1: NTP Control Message Header

Leap Indicator (LI): This is a two-bit integer that is set to b00 for control message requests and responses. The Leap Indicator value used at this position in most NTP modes is in the System Status Word provided in some control message responses.

Version Number (VN): This is a three-bit integer indicating a minimum NTP version number. NTP servers do not respond to control messages with an unrecognized version number. Requests may intentionally use a lower version number to enable interoperability with earlier versions of NTP. Responses carry the same version as the corresponding request.

Mode: This is a three-bit integer indicating the mode. The value 6 indicates an NTP control message.

Response Bit (R): Set to zero for commands, one for responses.

Error Bit (E): Set to zero for normal response, one for error response.

More Bit (M): Set to zero for last fragment, one for all others.

Operation Code (OpCode): This is a five-bit integer specifying the command function. Values currently defined include the following:

Code	Meaning
0	reserved
1	read status command/response
2	read variables command/response
3	write variables command/response
4	read clock variables command/response
5	write clock variables command/response
6	set trap address/port command/response
7	trap response
8	runtime configuration command/response
9	export configuration to file command/response
10	retrieve remote address stats command/response
11	retrieve ordered list command/response
12	request client-specific nonce command/response
13-30	reserved
31	unset trap address/port command/response

Sequence Number: This is a 16-bit integer indicating the sequence number of the command or response. Each request uses a different sequence number. Each response carries the same sequence number as its corresponding request. For asynchronous trap responses, the responder increments the sequence number by one for each response, allowing trap receivers to detect missing trap responses. The sequence number of each fragment of a multiple-datagram response carries the same sequence number, copied from the request.

Status: This is a 16-bit code indicating the current status of the system, peer or clock, with values coded as described in following sections.

Association ID: This is a 16-bit unsigned integer identifying a valid association, or zero for the system clock.

Offset: This is a 16-bit unsigned integer indicating the offset, in octets, of the first octet in the data area. The offset is set to zero in requests. Responses spanning multiple datagrams use a positive offset in all but the first datagram.

Count: This is a 16-bit unsigned integer indicating the length of the data field, in octets.

Data: This contains the message data for the command or response. The maximum number of data octets is 468.

Padding (optional): Contains zero to three octets with value zero, as needed to ensure the overall control message size is a multiple of 4 octets.

Authenticator (optional): When the NTP authentication mechanism is implemented, this contains the authenticator information defined in Appendix C of [RFC1305].

3. Status Words

Status words indicate the present status of the system, associations and clock. They are designed to be interpreted by network-monitoring programs and are in one of four 16-bit formats shown in Figure 2 and described in this section. System and peer status words are associated with responses for all commands except the read clock variables, write clock variables and set trap address/port commands. The association identifier zero specifies the system status word, while a nonzero identifier specifies a particular peer association. The status word returned in response to read clock variables and write clock variables commands indicates the state of the clock hardware and decoding software. A special error status word is used to report malformed command fields or invalid values.

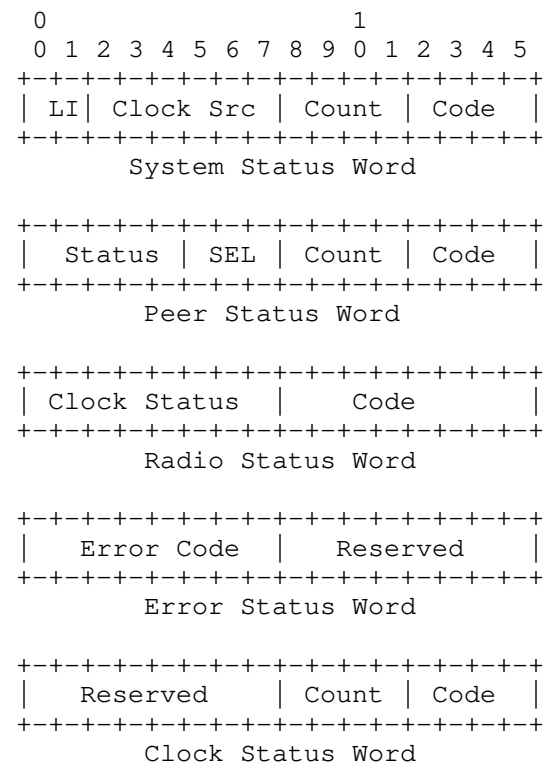

Figure 2: Status Word Formats

3.1. System Status Word

The system status word appears in the status field of the response to a read status or read variables command with a zero association identifier. The format of the system status word is as follows:

Leap Indicator (LI): This is a two-bit code warning of an impending leap second to be inserted/deleted in the last minute of the current day, with bit 0 and bit 1, respectively, coded as follows:

LI	Meaning
00	no warning
01	insert second after 23:59:59 of the current day
10	delete second 23:59:59 of the current day
11	unsynchronized

Clock Source (Clock Src): This is a six-bit integer indicating the current synchronization source, with values coded as follows:

Code	Meaning
0	unspecified or unknown
1	Calibrated atomic clock (e.g., PPS, HP 5061)
2	VLF (band 4) or LF (band 5) radio (e.g., OMEGA,, WWVB)
3	HF (band 7) radio (e.g., CHU, MSF, WWV/H)
4	UHF (band 9) satellite (e.g., GOES, GPS)
5	local net (e.g., DCN, TSP, DTS)
6	UDP/NTP
7	UDP/TIME
8	eyeball-and-wristwatch
9	telephone modem (e.g., NIST)
10-63	reserved

System Event Counter (Count): This is a four-bit integer indicating the number of system events occurring since the last time the System Event Code changed. Upon reaching 15, subsequent events with the same code are not counted.

System Event Code (Code): This is a four-bit integer identifying the latest system exception event, with new values overwriting previous values, and coded as follows:

Code	Meaning
0	unspecified
1	frequency correction (drift) file not available
2	frequency correction started (frequency stepped)
3	spike detected and ignored, starting stepout timer
4	frequency training started
5	clock synchronized
6	system restart
7	panic stop (required step greater than panic threshold)
8	no system peer
9	leap second insertion/deletion armed for the of the current month
10	leap second disarmed
11	leap second inserted or deleted
12	clock stepped (stepout timer expired)
13	kernel loop discipline status changed
14	leapseconds table loaded from file
15	leapseconds table outdated, updated file needed

3.2. Peer Status Word

A peer status word is returned in the status field of a response to a read status, read variables or write variables command and appears also in the list of association identifiers and status words returned by a read status command with a zero association identifier. The format of a peer status word is as follows:

Peer Status (Status): This is a five-bit code indicating the status of the peer determined by the packet procedure, with bits assigned as follows:

Peer Status bit	Meaning
0	configured (peer.config)
1	authentication enabled (peer.authenable)
2	authentication okay (peer.authentic)
3	reachability okay (peer.reach != 0)
4	broadcast association

Peer Selection (SEL): This is a three-bit integer indicating the status of the peer determined by the clock-selection procedure, with values coded as follows:

Sel	Meaning
0	rejected
1	discarded by intersection algorithm
2	discarded by table overflow (not currently used)
3	discarded by the cluster algorithm
4	included by the combine algorithm
5	backup source (with more than sys.maxclock survivors)
6	system peer (synchronization source)
7	PPS (pulse per second) peer

Peer Event Counter (Count): This is a four-bit integer indicating the number of peer exception events that occurred since the last time the peer event code changed. Upon reaching 15, subsequent events with the same code are not counted.

Peer Event Code (Code): This is a four-bit integer identifying the latest peer exception event, with new values overwriting previous values, and coded as follows:

Peer Event Code	Meaning
0	unspecified
1	association mobilized
2	association demobilized
3	peer unreachable (peer.reach was nonzero now zero)
4	peer reachable (peer.reach was zero now nonzero)
5	association restarted or timed out
6	no reply (only used with one-shot clock set command)
7	peer rate limit exceeded (kiss code RATE received)
8	access denied (kiss code DENY received)
9	leap second insertion/deletion at month's end armed by peer vote
10	became system peer (sys.peer)
11	reference clock event (see clock status word)
12	authentication failed
13	popcorn spike suppressed by peer clock filter register
14	entering interleaved mode
15	recovered from interleave error

3.3. Clock Status Word

There are two ways a reference clock can be attached to a NTP service host, as a dedicated device managed by the operating system and as a synthetic peer managed by NTP. As in the read status command, the association identifier is used to identify which one, zero for the system clock and nonzero for a peer clock. Only one system clock is supported by the protocol, although many peer clocks can be supported. A system or peer clock status word appears in the status field of the response to a read clock variables or write clock variables command. This word can be considered an extension of the system status word or the peer status word as appropriate. The format of the clock status word is as follows:

Reserved: An eight-bit integer that is ignored by requesters and zeroed by responders.

Count: This is a four-bit integer indicating the number of clock events that occurred since the last time the clock event code changed. Upon reaching 15, subsequent events with the same code are not counted.

Clock Code (Code): This is a four-bit integer indicating the current clock status, with values coded as follows:

Clock Status	Meaning
0	clock operating within nominals
1	reply timeout
2	bad reply format
3	hardware or software fault
4	propagation failure
5	bad date format or value
6	bad time format or value
7-15	reserved

3.4. Error Status Word

An error status word is returned in the status field of an error response as the result of invalid message format or contents. Its presence is indicated when the E (error) bit is set along with the response (R) bit in the response. It consists of an eight-bit integer coded as follows:

Error Status	Meaning
0	unspecified
1	authentication failure
2	invalid message length or format
3	invalid opcode
4	unknown association identifier
5	unknown variable name
6	invalid variable value
7	administratively prohibited
8-255	reserved

4. Commands

Commands consist of the header and optional data field shown in Figure 1. When present, the data field contains a list of identifiers or assignments in the form <<identifier>>[=<<value>>],<<identifier>>[=<<value>>],... where <<identifier>> is the ASCII name of a system or peer variable such as the ones specified in RFC 5905 and <<value>> is expressed as a decimal, hexadecimal or string constant in the syntax of the C programming language. Where no ambiguity exists, the "sys." or "peer." prefixes can be suppressed. Whitespace (ASCII nonprinting format effectors) can be added to improve readability for simple monitoring programs that do not reformat the data field. Internet addresses are represented as follows: IPv4 addresses are written in the form [n.n.n.n], where n is in decimal notation and the brackets are optional; IPv6 addresses are formulated based on the guidelines defined in [RFC5952]. Timestamps, including reference, originate, receive and transmit values, as well as the logical clock, are represented in units of seconds and fractions, preferably in hexadecimal notation. Delay, offset, dispersion and distance values are represented in units of milliseconds and fractions, preferably in decimal notation. All other values are represented as-is, preferably in decimal notation.

Implementations may define variables other than those described in RFC 5905. Called extramural variables, these are distinguished by the inclusion of some character type other than alphanumeric or "." in the name. For those commands that return a list of assignments in the response data field, if the command data field is empty, it is expected that all available variables defined in RFC 5905 will be included in the response. For the read commands, if the command data field is nonempty, an implementation may choose to process this field to individually select which variables are to be returned.

Commands are interpreted as follows:

Read Status (1): The command data field is empty or contains a list of identifiers separated by commas. The command operates in two ways depending on the value of the association identifier. If this identifier is nonzero, the response includes the peer identifier and status word. Optionally, the response data field may contain other information, such as described in the Read Variables command. If the association identifier is zero, the response includes the system identifier (0) and status word, while the data field contains a list of binary-coded pairs <<association identifier>> <<status word>>, one for each currently defined association.

Read Variables (2): The command data field is empty or contains a list of identifiers separated by commas. If the association identifier is nonzero, the response includes the requested peer identifier and status word, while the data field contains a list of peer variables and values as described above. If the association identifier is zero, the data field contains a list of system variables. If a peer has been selected as the synchronization source, the response includes the peer identifier and status word; otherwise, the response includes the system identifier (0) and status word.

Write Variables (3): The command data field contains a list of assignments as described above. The variables are updated as indicated. The response is as described for the Read Variables command.

Read Clock Variables (4): The command data field is empty or contains a list of identifiers separated by commas. The association identifier selects the system clock variables or peer clock variables in the same way as in the Read Variables command. The response includes the requested clock identifier and status word and the data field contains a list of clock variables and values, including the last timecode message received from the clock.

Write Clock Variables (5): The command data field contains a list of assignments as described above. The clock variables are updated as indicated. The response is as described for the Read Clock Variables command.

Set Trap Address/Port (6): The command association identifier, status and data fields are ignored. The address and port number for subsequent trap messages are taken from the source address and port of the control message itself. The initial trap counter for trap response messages is taken from the sequence field of the command. The response association identifier, status and data fields are not

significant. Implementations should include sanity timeouts which prevent trap transmissions if the monitoring program does not renew this information after a lengthy interval.

Trap Response (7): This message is sent when a system, peer or clock exception event occurs. The opcode field is 7 and the R bit is set. The trap counter is incremented by one for each trap sent and the sequence field set to that value. The trap message is sent using the IP address and port fields established by the set trap address/port command. If a system trap the association identifier field is set to zero and the status field contains the system status word. If a peer trap the association identifier field is set to that peer and the status field contains the peer status word. Optional ASCII-coded information can be included in the data field.

Configure (8): The command data is parsed and applied as if supplied in the daemon configuration file.

Save Configuration (9): Write a snapshot of the current configuration to the file name supplied as the command data. Further, the command is refused unless a directory in which to store the resulting files has been explicitly configured by the operator.

Read Most Recently Used (MRU) list (10): Retrieves records of recently seen remote addresses and associated statistics. Command data consists of name=value pairs controlling the selection of records, as well as a requestor-specific nonce previously retrieved using this command or opcode 12, Request Nonce. The response consists of name=value pairs where some names can appear multiple times using a dot followed by a zero-based index to distinguish them, and to associate elements of the same record with the same index. A new nonce is provided with each successful response.

Read ordered list (11): Retrieves a list ordered by IP address (IPv4 information precedes IPv6 information). If the command data is empty or the seven characters "ifstats", the associated statistics, status and counters for each local address are returned. If the command data is the characters "addr_restrictions" then the set of IPv4 remote address restrictions followed by the set of IPv6 remote address restrictions (access control lists) are returned. Other command data returns error code 5 (unknown variable name). Similar to Read MRU, response information uses zero-based indexes as part of the variable name preceding the equals sign and value, where each index relates information for a single address or network. This opcode requires authentication.

Request Nonce (12): Retrieves a 96-bit nonce specific to the requesting remote address, which is valid for a limited period.

Command data is not used in the request. The nonce consists of a 64-bit NTP timestamp and 32 bits of hash derived from that timestamp, the remote address, and salt known only to the server which varies between daemon runs. Inclusion of the nonce by a management agent demonstrates to the server that the agent can receive datagrams sent to the source address of the request, making source address "spoofing" more difficult in a similar way as TCP's three-way handshake.

Unset Trap (31): Removes the requesting remote address and port from the list of trap receivers. Command data is not used in the request. If the address and port are not in the list of trap receivers, the error code is 4, bad association.

5. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

6. Security Considerations

A number of security vulnerabilities have been identified with these control messages.

NTP's control query interface allows reading and writing of system, peer, and clock variables remotely from arbitrary IP addresses using commands mentioned in Section 4. Traditionally, overwriting these variables, but not reading them, requires authentication by default. However, this document argues that an NTP host must authenticate all control queries and not just ones that overwrite these variables. Alternatively, the host can use an access control list to explicitly list IP addresses that are allowed to control query the clients. These access controls are required for the following reasons:

- NTP as a Distributed Denial-of-Service (DDoS) vector. NTP timing query and response packets (modes 1-2, 3-4, 5) are usually short in size. However, some NTP control queries generate a very long packet in response to a short query. As such, there is a history of use of NTP's control queries, which exhibit such behavior, to perform DDoS attacks. These off-path attacks exploit the large size of NTP control queries to cause UDP-based amplification attacks (e.g., mode 7 monlist command generates a very long packet in response to a small query [CVE-DOS]). These attacks only use NTP as a vector for DoS attacks on other protocols, but do not affect the time service on the NTP host itself. To limit the

sources of these malicious commands, NTP server operators are recommended to deploy ingress filtering [RFC3704].

- Time-shifting attacks through information leakage/overwriting. NTP hosts save important system and peer state variables. An off-path attacker who can read these variables remotely can leverage the information leaked by these control queries to perform time-shifting and DoS attacks on NTP clients. These attacks do affect time synchronization on the NTP hosts. For instance,

  * In the client/server mode, the client stores its local time when it sends the query to the server in its xmt peer variable. This variable is used to perform TEST2 to non-cryptographically authenticate the server, i.e., if the origin timestamp field in the corresponding server response packet matches the xmt peer variable, then the client accepts the packet. An off-path attacker, with the ability to read this variable can easily spoof server response packets for the client, which will pass TEST2, and can deny service or shift time on the NTP client. The specific attack is described in [CVE-SPOOF].

  * The client also stores its local time when the server response is received in its rec peer variable. This variable is used for authentication in interleaved-pivot mode. An off-path attacker with the ability to read this state variable can easily shift time on the client by passing this test. This attack is described in [CVE-SHIFT].

- Fast-Scanning. NTP mode 6 control messages are usually small UDP packets. Fast-scanning tools like ZMap can be used to spray the entire (potentially reachable) Internet with these messages within hours to identify vulnerable hosts. To make things worse, these attacks can be extremely low-rate, only requiring a control query for reconnaissance and a spoofed response to shift time on vulnerable clients.

- The mode 6 and 7 messages are vulnerable to replay attacks [CVE-Replay]. If an attacker observes mode 6/7 packets that modify the configuration of the server in any way, the attacker can apply the same change at any time later simply by sending the packets to the server again. The use of the nonce (Request Nonce command) provides limited protection against replay attacks.

NTP best practices recommend configuring NTP with the no-query parameter. The no-query parameter blocks access to all remote control queries. However, sometimes the hosts do not want to block all queries and want to give access for certain control queries remotely. This could be for the purpose of remote management and

configuration of the hosts in certain scenarios. Such hosts tend to use firewalls or other middleboxes to blacklist certain queries within the network.

Significantly fewer hosts respond to mode 7 monlist queries as compared to other control queries because it is a well-known and exploited control query. These queries are likely blocked using blacklists on firewalls and middleboxes rather than the no-query option on NTP hosts. The remaining control queries that can be exploited likely remain out of the blacklist because they are undocumented in the current NTP specification [RFC5905].

This document describes all of the mode 6 control queries allowed by NTP and can help administrators make informed decisions on security measures to protect NTP devices from harmful queries and likely make those systems less vulnerable. Regardless of which mode 6 commands an administrator may elect to allow, remote access to this facility needs to be protected from unauthorized access (e.g., strict ACLs).

7. Contributors

Dr. David Mills specified the vast majority of the mode 6 commands during the development of RFC 1305 [RFC1305] and deserves the credit for their existence and use.

8. Acknowledgements

Tim Plunkett created the original version of this document. Aanchal Malhotra provided the initial version of the Security Considerations section.

Karen O'Donoghue, David Hart, Harlan Stenn, and Philip Chimento deserve credit for portions of this document due to their earlier efforts to document these commands.

Miroshav Lichvar, Ulrich Windl, Dieter Sibold, J Ignacio Alvarez-Hamelin, and Alex Campbell provided valuable comments on various versions of this document.

9. References

9.1. Normative 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>.

[RFC3704] Baker, F. and P. Savola, "Ingress Filtering for Multihomed Networks", BCP 84, RFC 3704, DOI 10.17487/RFC3704, March 2004, <https://www.rfc-editor.org/info/rfc3704>.

[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>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <https://www.rfc-editor.org/info/rfc5952>.

9.2. Informative References

[CVE-DOS] NIST National Vulnerability Database, "CVE-2013-5211, https://nvd.nist.gov/vuln/detail/CVE-2013-5211", January 2014.

[CVE-Replay] NIST National Vulnerability Database, "CVE-2015-8140, https://nvd.nist.gov/vuln/detail/CVE-2015-8140", January 2015.

[CVE-SHIFT] NIST National Vulnerability Database, "CVE-2016-1548, https://nvd.nist.gov/vuln/detail/CVE-2016-1548", January 2017.

[CVE-SPOOF] NIST National Vulnerability Database, "CVE-2015-8139, https://nvd.nist.gov/vuln/detail/CVE-2015-8139", January 2017.

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

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <https://www.rfc-editor.org/info/rfc2460>.

Appendix A. NTP Remote Facility Message Format

The format of the NTP Remote Facility Message header, which immediately follows the UDP header, is shown in Figure 3. Following

is a description of its fields. Bit positions marked as zero are reserved and should always be transmitted as zero.

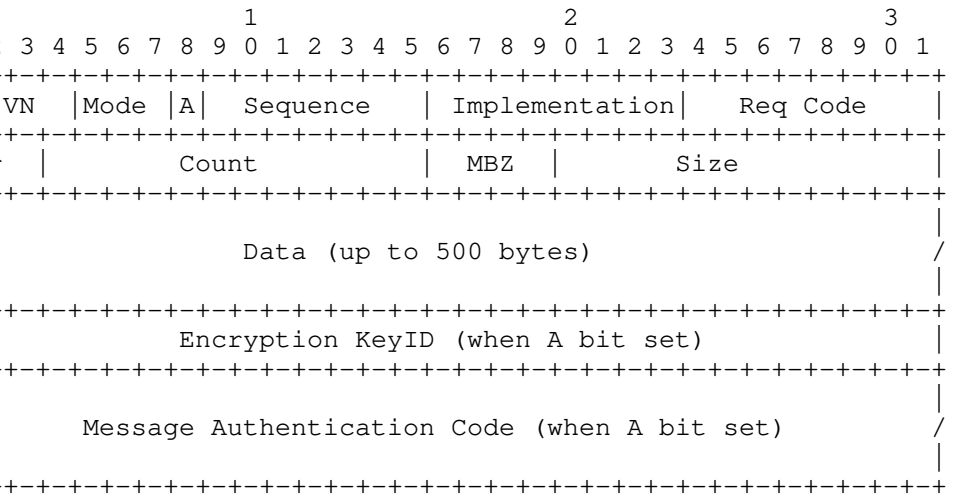

Figure 3: NTP Remote Facility Message Header

Response Bit (R) : Set to 0 if the packet is a request. Set to 1 if the packet is a response.

More Bit (M) : Set to 0 if this is the last packet in a response, otherwise set to 1 in responses requiring more than one packet.

Version Number (VN) : Set to the version number of the NTP daemon.

Mode : Set to 7 for Remote Facility messages.

Authenticated Bit (A) : If set to 1, this packet contains authentication information.

Sequence : For a multi-packet response, this field contains the sequence number of this packet. Packets in a multi-packet response are numbered starting with 0. The More Bit is set to 1 for all packets but the last.

Implementation : The version number of the implementation that defined the request code used in this message. An implementation number of 0 is used for a Request Code supported by all versions of the NTP daemon. The value 255 is reserved for future extensions.

Request Code (Req Code) : An implementation-specific code which specifies the operation being requested. A Request Code definition includes the format and semantics of the data included in the packet.

Error (Err) : Set to 0 for a request. For a response, this field contains an error code relating to the request. If the Error is non-zero, the operation requested wasn't performed.

0 - no error

1 - incompatible implementation number

2 - unimplemented request code

3 - format error

4 - no data available

7 - authentication failure

Count : The number of data items in the packet. Range is 0 to 500.

Must Be Zero (MBZ) : A reserved field set to 0 in requests and responses.

Size : The size of each data item in the packet. Range is 0 to 500.

Data : A variable-sized field containing request/response data. For requests and responses, the size in octets must be greater than or equal to the product of the number of data items (Count) and the size of a data item (Size). For requests, the data area is exactly 40 octets in length. For responses, the data area will range from 0 to 500 octets, inclusive.

Encryption KeyID : A 32-bit unsigned integer used to designate the key used for the Message Authentication Code. This field is included only when the A bit is set to 1.

Message Authentication Code : An optional Message Authentication Code defined by the version of the NTP daemon indicated in the Implementation field. This field is included only when the A bit is set to 1.

Author's Address

Brian Haberman (editor)
JHU

Email: brian@innovationslab.net

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: March 2, 2021

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

August 29, 2020

Roughtime
draft-ietf-ntp-roughtime-03

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)

[I-D.ietf-ntp-using-nts-for-ntp] fail 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].

More specifically,

- 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.

Chaining multiple servers: For subsequent requests, the client generates a new nonce by hashing the reply from the previous server with a random value (a blind). This proves that the nonce was created after the reply from the previous server. It sends the new nonce in a request to the next server and receives a response that includes a signature covering the nonce.

Cryptographic proof of misbehavior: If the time from the second server is before the first, then the client has proof that at least one of the servers is misbehaving; the reply from the second server implicitly shows that it was created later because of the way that the client constructed the nonce. If the time from the second server is too far in the future, the client can contact the first server again with a new nonce generated from the second server's response and get a signature that was provably created afterwards, but with an earlier timestamp.

With only two servers, the client can end up with proof that something is wrong, but no idea what the correct time is. But with half a dozen or more independent servers, the client will end up with chain of proof of any server's misbehavior, signed by several others, and (presumably) enough accurate replies to establish what the correct time is. Furthermore, this proof may be validated by third parties ultimately leading to a revocation of trust in the misbehaving server.

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.

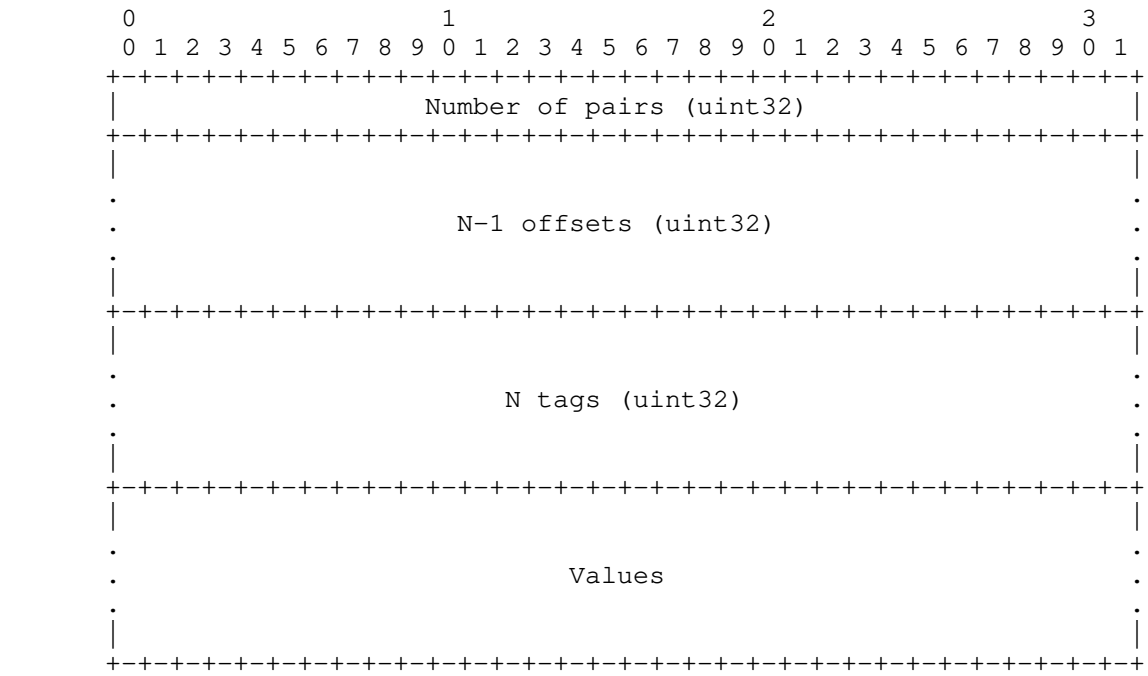

Figure 1: Roughtime Message Format

5.1. Data Types

5.1.1. int32

An int32 is a 32 bit signed integer. It is serialized in sign-magitude 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 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.

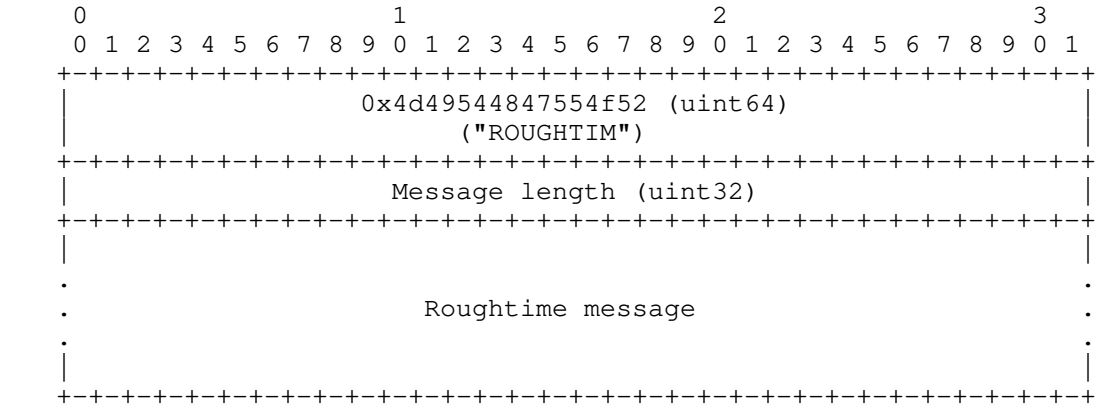

Figure 2: Roughtime Packet Format

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 is a Roughtime message with the tags NONC and VER. Over a UDP connection the size of the request message SHOULD be at least 1024 bytes. To attain this size the PAD tag SHOULD be added to the message. Tags other than NONC and VER SHOULD be ignored by the server. Responding to requests shorter than 1024 bytes is OPTIONAL and servers MUST NOT send responses larger than the requests they are replying to.

The value of the NONC tag is a 64 byte nonce. It SHOULD be generated by hashing a previous Roughtime response message together with a blind as described in Section 8. If no previous responses are avaiable to the client, the nonce SHOULD be generated at 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.

The PAD tag SHOULD be used by clients to ensure their request messages are at least 1024 bytes in size. Its value SHOULD be all zeros.

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 value represents the MJD of a past or future leap second event. Positive values represent the addition of a second at the indicated date and negative values represent the removal of a second at the indicated (negative) date. The first item in the list MUST be the last (past or future) leap second event that the server knows about. The leap second events MUST be sorted in reverse chronological order. 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. Cheater Detection

A chain of responses is a series of responses where the SHA-512/256 hash of the preceding response H, is concatenated with a 64 byte blind X, and then SHA-512/256(H, X) is the nonce used in the subsequent response. These may be represented as an array of objects in JavaScript Object Notation (JSON) format [RFC8259] where each object may have keys "blind" and "response_packet". Packet has the Base64 [RFC4648] encoded bytes of the packet and blind is the Base64 encoded blind used for the next nonce. The last packet needs no blind.

A pair of responses (r_1, r_2) is invalid if MIDP_1-RADI_1 > MIDP_2+RADI_2. A chain of longer length is invalid if for any i, j such that i < j, (r_i, r_j) is an invalid pair.

Invalidity of a chain is proof that causality has been violated if all servers were reporting correct time. An invalid chain where all individual responses are valid is cryptographic proof of malfeasance of at least one server: if all servers had the correct time in the chain, causality would imply that MIDP_1-RADI_1 < MIDP_2+RADI_2.

In conducting the comparison of timestamps one must know the length of a day and hence have historical leap second data for the days in question. However if violations are greater then a second the loss of leap second data doesn’t impede their detection.

9. 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.

10. 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=
```

11. Trust Anchors and Policies

A trust anchor is any distributor of a list of trusted servers. It is RECOMMENDED that trust anchors subscribe to a common public forum where evidence of malfeasance may be shared and discussed. Trust anchors SHOULD subscribe to a zero-tolerance policy: any generation of incorrect timestamps will result in removal. To enable this trust anchors SHOULD list a wide variety of servers so the removal of a server does not result in operational issues for clients. Clients SHOULD attempt to detect malfeasance and have a way to report it to trust anchors.

Because only a single Roughtime server is required for successful synchronization, Roughtime does not have the incentive problems that have prevented effective enforcement of discipline on the web PKI. We expect that some clients will aggressively monitor server behavior.

12. Acknowledgements

Marcus Dansarie contributed many fruitful ideas. 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.

13. IANA Considerations

13.1. 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

13.2. Roughtime Version Registry

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

Version (REQUIRED): a 32-bit unsigned integer

Reference (REQUIRED): the description of the version

The policy for allocation of new entries SHOULD be IETF consensus. Versions with the high bit set are reserved.

The initial contents of this registry shall be as follows:

Version	Reference
1	[[this memo]]

13.3. 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]]
0x434e4f48	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]]

14. Security Considerations

Since the only supported signature scheme, Ed25519, is not quantum resistant, this protocol 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.

15. 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 from previous responses without using a blind can enable tracking of clients as they move between networks.

16. References

16.1. 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>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[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] NIST, "Secure Hash Standard", FIPS 180-4, August 2015.

16.2. 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>.

[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-25 (work in progress), March 2020.

[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>.

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 [I-D.ietf-ntp-using-nts-for-ntp]

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

NTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 14, 2021

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

July 13, 2020

A YANG Data Model for NTP
draft-ietf-ntp-yang-data-model-09

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 January 14, 2021.

Table of Contents

1. Introduction

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

The data model convers 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 parameters of per-interface. 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 [RFC8342]. NTP current state and statistics are also maintained in the operational state. Additionally, the operational state also include the associations 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]
ianach	iana-crypt-hash	[RFC7317]
key-chain	ietf-key-chain	[RFC8177]
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. Refrences in the Model

Following documents are refrenced 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]

Table 2: Refrences 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
     |  +--rw auth-enabled?          boolean
     |  +--rw authentication-keys* [key-id]
     |     +--rw key-id       uint32
     |     +--...
     +--rw access-rules
     |  +--rw access-rule* [access-mode]
     |     +--rw access-mode    access-mode
     |     +--rw acl?           -> /acl:acls/acl/name
     +--ro clock-state
     |  +--ro system-status
     |     +--ro clock-state                 ntp-clock-status
     |     +--ro clock-stratum               ntp-stratum
     |     +--ro clock-refid                 union
     |     +--...
     +--rw unicast-configuration* [address type]
     |  +--rw address           inet:host
     |  +--rw type              unicast-configuration-type
     |  +--...
     +--ro associations* [address local-mode isconfigured]
     |  +--...
     |  +--ro ntp-statistics
     |     +--...
     +--rw interfaces
     |  +--rw interface* [name]
     |     +--rw name                if:interface-ref
     |     +--rw broadcast-server!
     |     |  +--...
     |     +--rw broadcast-client!
     |     +--rw multicast-server* [address]
     |     |  +--rw address
     |     |  |       rt-types:ip-multicast-group-address
     |     |  +--...
     |     +--rw multicast-client* [address]
     |     |  +--rw address    rt-types:ip-multicast-group-address
     |     +--rw manycast-server* [address]
     |     |  +--rw address    rt-types:ip-multicast-group-address
     |     +--rw manycast-client* [address]
     |        +--rw address
     |        |       rt-types:ip-multicast-group-address
     |        +--...
     +--ro ntp-statistics
        +--...
```

The full data model tree for the YANG module "ietf-ntp" is represented as -

```
module: ietf-ntp
  +--rw ntp!
     +--rw port?                    inet:port-number {ntp-port}?
     +--rw refclock-master!
     |  +--rw master-stratum?   ntp-stratum
     +--rw authentication
     |  +--rw auth-enabled?          boolean
     |  +--rw authentication-keys* [key-id]
     |     +--rw key-id       uint32
     |     +--rw algorithm?   identityref
     |     +--rw key?         ianach:crypt-hash
     |     +--rw istrusted?   boolean
     +--rw access-rules
     |  +--rw access-rule* [access-mode]
     |     +--rw access-mode    access-mode
     |     +--rw acl?           -> /acl:acls/acl/name
     +--ro clock-state
     |  +--ro system-status
     |     +--ro clock-state                 ntp-clock-status
     |     +--ro clock-stratum               ntp-stratum
     |     +--ro clock-refid                 union
     |     +--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             uint8
     |     +--ro clock-offset?               decimal64
     |     +--ro root-delay?                 decimal64
     |     +--ro root-dispersion?            decimal64
     |     +--ro reference-time?             yang:date-and-time
     |     +--ro sync-state                  ntp-sync-state
     +--rw unicast-configuration* [address type]
     |  +--rw address           inet:host
     |  +--rw type              unicast-configuration-type
     |  +--rw 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?          ntp-minpoll
        |  +--rw maxpoll?          ntp-maxpoll
        |  +--rw port?             inet:port-number {ntp-port}?
        |  +--rw version?          ntp-version
        +--ro associations* [address local-mode isconfigured]
        |  +--ro address           inet:host
        |  +--ro local-mode        association-mode
        |  +--ro isconfigured      boolean
        |  +--ro stratum?          ntp-stratum
        |  +--ro refid?            union
        |  +--ro authentication?
        |  |       -> /ntp/authentication/authentication-keys/key-id
        |  +--ro prefer?           boolean
        |  +--ro peer-interface?   if:interface-ref
        |  +--ro minpoll?          ntp-minpoll
        |  +--ro maxpoll?          ntp-maxpoll
        |  +--ro port?             inet:port-number {ntp-port}?
        |  +--ro version?          ntp-version
        |  +--ro reach?            uint8
        |  +--ro unreach?          uint8
        |  +--ro poll?             uint8
        |  +--ro now?              uint32
        |  +--ro offset?           decimal64
        |  +--ro delay?            decimal64
        |  +--ro dispersion?       decimal64
        |  +--ro originate-time?   yang:date-and-time
        |  +--ro receive-time?     yang:date-and-time
        |  +--ro transmit-time?    yang:date-and-time
        |  +--ro input-time?       yang:date-and-time
        |  +--ro ntp-statistics
        |     +--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!
        |     |  +--rw ttl?              uint8
        |     |  +--rw authentication
        |     |  |  +--rw (authentication-type)?
        |     |  |     +--:(symmetric-key)
        |     |  |        +--rw key-id?   leafref
        |     |  +--rw minpoll?          ntp-minpoll
        |     |  +--rw maxpoll?          ntp-maxpoll
        |     |  +--rw port?             inet:port-number {ntp-port}?
        |     |  +--rw version?          ntp-version
```

```
|      +--rw broadcast-client!
|      +--rw multicast-server* [address]
|      |  +--rw address
|      |  |       rt-types:ip-multicast-group-address
|      |  +--rw ttl?              uint8
|      |  +--rw authentication
|      |  |  +--rw (authentication-type)?
|      |  |     +--:(symmetric-key)
|      |  |        +--rw key-id?   leafref
|      |  +--rw minpoll?          ntp-minpoll
|      |  +--rw maxpoll?          ntp-maxpoll
|      |  +--rw port?             inet:port-number {ntp-port}?
|      |  +--rw version?          ntp-version
|      +--rw multicast-client* [address]
|      |  +--rw address    rt-types:ip-multicast-group-address
|      +--rw manycast-server* [address]
|      |  +--rw address    rt-types:ip-multicast-group-address
|      +--rw manycast-client* [address]
|         +--rw address
|         |       rt-types:ip-multicast-group-address
|         +--rw authentication
|         |  +--rw (authentication-type)?
|         |     +--:(symmetric-key)
|         |        +--rw key-id?   leafref
|         +--rw ttl?              uint8
|         +--rw minclock?         uint8
|         +--rw maxclock?         uint8
|         +--rw beacon?           uint8
|         +--rw minpoll?          ntp-minpoll
|         +--rw maxpoll?          ntp-maxpoll
|         +--rw port?             inet:port-number {ntp-port}?
|         +--rw version?          ntp-version
|
+--ro ntp-statistics
   +--ro packet-sent?        yang:counter32
   +--ro packet-sent-fail?   yang:counter32
   +--ro packet-received?    yang:counter32
   +--ro packet-dropped?     yang:counter32
```

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 supports per-interface configurations 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

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 (such as peer or server; refer access-mode) and attach different acl-rule to each. For this, the YANG module allow such configuration via /ntp/access-rules. The access-rule itself is configured via [RFC8519].

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 32-bits unsigned integer that MUST be configured on the NTP peers before the authentication could be used. For this reason, this YANG modules allow such configuration via /ntp/authentication/authentication-keys/. Further at the time of configuration of NTP association (for example unicast-server), the key-id is specefied.

7. NTP YANG Module

```
<CODE BEGINS> file "ietf-ntp@2020-07-13.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 iana-crypt-hash {
      prefix ianach;
      reference
        "RFC 7317: A YANG Data Model for System Management";
    }
    import ietf-key-chain {
      prefix key-chain;
      reference
        "RFC 8177: YANG Data Model for Key Chains";
    }
    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: ntpwg@lists.ntp.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) 2020 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 2020-07-13 {
      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";
    }

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

    typedef ntp-minpoll {
      type uint8 {
        range "4..17";
      }
      default "6";
      description
        "The minimum poll exponent for this NTP association.";
      reference
        "RFC 5905: Network Time Protocol Version 4: Protocol and
         Algorithms Specification";
    }

    typedef ntp-maxpoll {
      type uint8 {
        range "4..17";
      }
      default "10";
      description
        "The maximum poll exponent for this NTP association.";
      reference
        "RFC 5905: Network Time Protocol Version 4: Protocol and
         Algorithms Specification";
    }

    typedef access-mode {
      type enumeration {
        enum peer {
          value 0;
          description
            "Enables the full access authority. Both time
             request and control query can be performed
             on the local NTP service, and the local clock
             can be synchronized with the remote server.";
        }
        enum server {
          value 1;
```

```
          description
            "Enables the server access and query.
             Both time requests and control query can be
             performed on the local NTP service, but the
             local clock cannot be synchronized with the
             remote server.";
        }
        enum synchronization {
          value 2;
          description
            "Enables the server to access.
             Only time request can be performed on the
             local NTP service.";
        }
        enum query {
          value 3;
          description
            "Enables the maximum access limitation.
             Control query can be performed only on the
             local NTP service.";
        }
      }
      description
        "This defines NTP access modes.";
    }

    typedef unicast-configuration-type {
      type enumeration {
        enum server {
          value 0;
          description
            "Use client association mode. This device
             will not provide synchronization to the
             configured NTP server.";
        }
        enum peer {
          value 1;
          description
            "Use symmetric active association mode.
             This device may provide synchronization
             to the configured NTP server.";
        }
      }
      description
        "This defines NTP unicast mode of operation.";
    }

    typedef association-mode {
```

```
      type enumeration {
        enum client {
          value 0;
          description
            "Use client association mode(mode 3).
             This device will not provide synchronization
             to the configured NTP server.";
        }
        enum active {
          value 1;
          description
            "Use symmetric active association mode(mode 1).
             This device may synchronize with its NTP peer,
             or provide synchronization to configured NTP peer.";
        }
        enum passive {
          value 2;
          description
            "Use symmetric passive association mode(mode 2).
             This device has learned this association dynamically.
             This device may synchronize with its NTP peer.";
        }
        enum broadcast {
          value 3;
          description
            "Use broadcast mode(mode 5).
             This mode defines that its either working
             as broadcast-server or multicast-server.";
        }
        enum broadcast-client {
          value 4;
          description
            "This mode defines that its either working
             as broadcast-client or multicast-client.";
        }
      }
      description
        "The NTP association modes.";
    }

    typedef ntp-clock-status {
      type enumeration {
        enum synchronized {
          value 0;
          description
            "Indicates that the local clock has been
             synchronized with an NTP server or
             the reference clock.";
```

```
        }
        enum unsynchronized {
          value 1;
          description
            "Indicates that the local clock has not been
             synchronized with any NTP server.";
        }
      }
      description
        "This defines NTP clock status.";
    }

    typedef ntp-sync-state {
      type enumeration {
        enum clock-not-set {
          value 0;
          description
            "Indicates the clock is not updated.";
        }
        enum freq-set-by-cfg {
          value 1;
          description
            "Indicates the clock frequency is set by
             NTP configuration.";
        }
        enum clock-set {
          value 2;
          description
            "Indicates the clock is set.";
        }
        enum freq-not-determined {
          value 3;
          description
            "Indicates the clock is set but the frequency
             is not determined.";
        }
        enum clock-synchronized {
          value 4;
          description
            "Indicates that the clock is synchronized";
        }
        enum spike {
          value 5;
          description
            "Indicates a time difference of more than 128
             milliseconds is detected between NTP server
             and client clock. The clock change will take
             effect in XXX seconds.";
```

```
        }
      }
      description
        "This defines NTP clock sync states.";
    }

    /* features */

    feature ntp-port {
      description
        "Support for NTP port configuration";
      reference
        "RFC 5905: Network Time Protocol Version 4: Protocol and
         Algorithms Specification";
    }

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

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

    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";
    }

    feature broadcast-server {
      description
        "Support for broadcast server";
      reference
        "RFC 5905: Network Time Protocol Version 4: Protocol and
         Algorithms Specification";
    }
```

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

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

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

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

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

    /* Groupings */

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

```
          range "1..max";
        }
        description
          "Authentication key identifier.";
      }
      leaf algorithm {
        type identityref {
          base key-chain:crypto-algorithm;
        }
        description
          "Authentication algorithm.";
      }
      leaf key {
        nacm:default-deny-all;
        type ianach:crypt-hash;
        description
          "The key";
      }
      leaf istrusted {
        type boolean;
        description
          "Key-id is trusted or not";
      }
      reference
        "RFC 5905: Network Time Protocol Version 4: Protocol and
         Algorithms Specification";
    }

    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 packet-sent {
        type yang:counter32;
        description
          "The total number of NTP packets delivered to the
           transport service by this NTP entity for this
           association.
           Discountinuities in the value of this counter can occur
           upon cold start or reinitialization of the NTP entity, the
           management system and at other times as indicated by
           discontinuities in the value of sysUpTime.";
      }
      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.
           Discountinuities in the value of this counter can occur
           upon cold start or reinitialization of the NTP entity, the
           management system and at other times as indicated by
           discontinuities in the value of sysUpTime.";
      }
      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.
           Discountinuities in the value of this counter can occur
           upon cold start or reinitialization of the NTP entity, the
           management system and at other times as indicated by
           discontinuities in the value of sysUpTime.";
      }
    }

    grouping common-attributes {
      description
        "NTP common attributes for configuration.";
      leaf minpoll {
        type ntp-minpoll;
        description
```

```
          "The minimum poll interval used in this association.";
      }
      leaf maxpoll {
        type ntp-maxpoll;
        description
          "The maximum poll interval used in this association.";
      }
      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.";
      }
      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 {
      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.";
      }
      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 {
        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 {
        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 access-mode;
            description
              "NTP access mode. The defination of each possible values:
               peer(0): Both time request and control query can be
               performed.
               server(1): Enables the server access and query.
               synchronization(2): Enables the server access only.
               query(3): 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";
        }
      }
      container clock-state {
        config false;
        description
          "Clock operational state of the NTP.";
        container system-status {
          description
            "System status of NTP.";
          leaf clock-state {
            type ntp-clock-status;
            mandatory true;
```

```
            description
              "The state of system clock. The definition of each
               possible value is:
               synchronized(0): Indicates local clock is synchronized.
               unsynchronized(1): Indicates local clock is not
               synchronized.";
          }
          leaf clock-stratum {
            type ntp-stratum;
            mandatory true;
            description
              "The NTP entity's own stratum value. Should be a stratum
               of syspeer + 1 (or 16 if no syspeer).";
            reference
              "RFC 5905: Network Time Protocol Version 4: Protocol and
               Algorithms Specification";
          }
          leaf clock-refid {
            type union {
              type inet:ipv4-address;
              type binary {
                length "4";
              }
              type string {
                length "4";
              }
            }
            mandatory true;
            description
              "IPv4 address or first 32 bits of the MD5 hash of
               the IPv6 address or reference clock of the peer to
               which clock is synchronized.";
            reference
              "RFC 5905: Network Time Protocol Version 4: Protocol and
               Algorithms Specification";
          }
          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.";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification";
        }
        leaf actual-freq {
          type decimal64 {
            fraction-digits 4;
          }
          units "Hz";
          mandatory true;
          description
            "The actual frequency of the
             local clock.";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification";
        }
        leaf clock-precision {
          type uint8;
          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";
        }
        leaf clock-offset {
          type decimal64 {
            fraction-digits 3;
          }
          units "milliseconds";
          description
            "The time offset to the current selected reference time
             source e.g., '0.032' or '1.232'.";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification";
        }
        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";
          }
          leaf root-dispersion {
            type decimal64 {
              fraction-digits 3;
            }
            units "milliseconds";
            description
              "The dispersion between the local clock
               and the root clock, e.g., '6.927'.";
            reference
              "RFC 5905: Network Time Protocol Version 4: Protocol and
               Algorithms Specification";
          }
          leaf reference-time {
            type yang:date-and-time;
            description
              "The reference timestamp.";
          }
          leaf sync-state {
            type ntp-sync-state;
            mandatory true;
            description
              "The synchronization status of
               the local clock.";
          }
        }
      }
      list unicast-configuration {
        key "address type";
        description
          "List of NTP unicast-configurations.";
        leaf address {
          type inet:host;
          description
            "Address of this association.";
        }
        leaf type {
          type unicast-configuration-type;
          description
            "Use client association mode.  This device
             will not provide synchronization to the
             configured NTP server.";
        }
        container 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";
        }
        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";
        }
        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 is 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:host;
           description
             "The address of this association. Represents the IP
              address of a unicast/multicast/broadcast address.";
         }
         leaf local-mode {
           type 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";
         }
         leaf refid {
           type union {
             type inet:ipv4-address;
             type binary {
               length "4";
             }
             type string {
               length "4";
```

```
            }
          }
          description
            "The refclock driver ID, if available.
             -- a refclock driver ID like '127.127.1.0' for local clock
               sync
             -- uni/multi/broadcast associations will look like
               '20.1.1.1'
             -- sync with primary source will look like 'DCN', 'NIST',
               'ATOM'";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification";
        }
        leaf 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";
        }
        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";
        }
        leaf poll {
          type uint8;
          units "seconds";
          description
            "The polling interval for current association";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification";
        }
        leaf now {
          type uint32;
          units "seconds";
          description
            "The time since the NTP packet was
             not received or last synchronized.";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification";
        }
        leaf offset {
          type decimal64 {
            fraction-digits 3;
          }
          units "milliseconds";
          description
            "The offset between the local clock
             and the peer clock, e.g., '0.032' or '1.232'";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification";
        }
        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";
        }
        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";
        }
        leaf originate-time {
          type yang:date-and-time;
          description
            "This is the local time, in timestamp format,
             when latest NTP packet was sent to peer(T1).";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification";
        }
        leaf receive-time {
          type yang:date-and-time;
          description
            "This is the local time, in timestamp format,
             when latest NTP packet arrived at peer(T2).
             If the peer becomes unreachable the value is set to zero.";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification";
        }
        leaf transmit-time {
          type yang:date-and-time;
          description
            "This is the local time, in timestamp format,
             at which the NTP packet departed the peer(T3).
             If the peer becomes unreachable the value is set to zero.";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification";
        }
        leaf input-time {
          type yang:date-and-time;
          description
            "This is the local time, in timestamp format,
```

```
            when the latest NTP message from the peer arrived(T4).
            If the peer becomes unreachable the value is set to zero.";
          reference
            "RFC 5905: Network Time Protocol Version 4: Protocol and
            Algorithms Specification";
        }
        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 {
            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.";
            }
            container authentication {
              description
                "Authentication used for this association.";
              uses authentication;
            }
            uses common-attributes {
              description
                "Common attribute like port, version, min and
                 max poll.";
            }
            reference
```

```
            "RFC 5905: Network Time Protocol Version 4: Protocol and
             Algorithms Specification";
          }
          container 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";
          }
          list 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.";
            }
            container authentication {
              description
                "Authentication used for this association.";
              uses authentication;
            }
            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";
          }
          list 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.";
        }
      }
      list 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";
      }
      list 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 {
          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.";
        }
        leaf minclock {
          type uint8;
          description
            "The minimum manycast survivors in this
             association.";
        }
        leaf maxclock {
          type uint8;
          description
            "The maximum manycast candidates in this
```

```
                association.";
            }
            leaf beacon {
              type uint8;
              description
                "The maximum interval between beacons in this
                 association.";
            }
            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";
          }
        }
      }
      container ntp-statistics {
        config false;
        description
          "Total NTP packet statistics.";
        uses statistics {
          description
            "NTP send and receive packet statistics.";
        }
      }
    }
  }
```

<CODE ENDS>

8. Usage Example

This section include examples for illustration purporses.

8.1. 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

```
    <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>server</type>
            <prefer>true</prefer>
            <version>4</version>
            <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.

This example is for retriving unicast configurations -

```
<get>
  <filter type="subtree">
  <sys:ntp xmlns:sys="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <sys:unicast-configuration>
      </sys:unicast-configuration>
  </sys: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>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>
          <version>4</version>
          <stratum>9</stratum>
          <refid>20.1.1.1</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>
```

8.2. 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">
    <sys:ntp xmlns:sys="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <sys:refclock-master>
      </sys:refclock-master>
    </sys: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>
```

8.3. Authentication configuration

This example describes how to enable authentication and configure trusted authentication key 10 with mode as md5 and key as 'abcd' -

```
<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>md5</algorithm>
          <key>abcd</key>
          <istrusted>true</istrusted>
        </authentication-keys>
      </authentication>
    </ntp>
  </config>
</edit-config>
```

This example describes how to get authentication related configuration -

```
<get>
  <filter type="subtree">
    <sys:ntp xmlns:sys="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <sys:authentication>
      </sys:authentication>
    </sys: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>md5</algorithm>
        <key>abcd</key>
        <istrusted>true</istrusted>
      </authentication-keys>
    </authentication>
  </ntp>
</data>
```

8.4. 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>
          <acl>2000</acl>
        </access-rule>
      </access-rules>
    </ntp>
  </config>
</edit-config>
```

This example describes how to get access related configuration -

```
<get>
  <filter type="subtree">
    <sys:ntp xmlns:sys="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <sys:access-rules>
      </sys:access-rules>
    </sys: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>
        <acl>2000</acl>
      </access-rule>
    </access-rules>
  </ntp>
</data>
```

8.5. Multicast configuration

This example describes how to configure multicast-server with address as "224.1.1.1", 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>
          <multicast-server>
            <address>224.1.1.1</address>
            <authentication>
              <symmetric-key>
                <key-id>10</key-id>
              </symmetric-key>
            </authentication>
            <port>1025</port>
          </multicast-server>
        </interface>
      </interfaces>
    </ntp>
  </config>
</edit-config>
```

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

```
<get>
  <filter type="subtree">
    <sys:ntp xmlns:sys="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <sys:interfaces>
        <sys:interface>
          <sys:multicast-server>
          </sys:multicast-server>
        </sys:interface>
      </sys:interfaces>
    </sys: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>224.1.1.1</address>
          <ttl>224.1.1.1</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 "224.1.1.1" -

```
<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>224.1.1.1</address>
          </multicast-client>
        </interface>
      </interfaces>
    </ntp>
  </config>
</edit-config>
```

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

```
<get>
  <filter type="subtree">
    <sys:ntp xmlns:sys="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <sys:interfaces>
        <sys:interface>
          <sys:multicast-client>
          </sys:multicast-client>
        </sys:interface>
      </sys:interfaces>
    </sys: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>224.1.1.1</address>
        </multicast-client>
      </interface>
    </interfaces>
  </ntp>
</data>
```

8.6. Manycast configuration

This example describes how to configure manycast-client with address as "224.1.1.1", 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>224.1.1.1</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">
    <sys:ntp xmlns:sys="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <sys:interfaces>
        <sys:interface>
          <sys:manycast-client>
          </sys:manycast-client>
        </sys:interface>
      </sys:interfaces>
    </sys: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>224.1.1.1</address>
          <authentication>
            <symmetric-key>
              <key-id>10</key-id>
            </symmetric-key>
          </authentication>
          <ttl>255</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 "224.1.1.1" -

```
<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>224.1.1.1</address>
          </manycast-server>
        </interface>
      </interfaces>
    </ntp>
  </config>
</edit-config>
```

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

```
<get>
  <filter type="subtree">
    <sys:ntp xmlns:sys="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <sys:interfaces>
        <sys:interface>
          <sys:manycast-server>
          </sys:manycast-server>
        </sys:interface>
      </sys:interfaces>
    </sys: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>224.1.1.1</address>
        </manycast-server>
      </interface>
    </interfaces>
  </ntp>
</data>
```

8.7. Clock state

This example describes how to get clock current state -

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

8.8. Get all association

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

```
   <get>
     <filter type="subtree">
       <sys:ntp xmlns:sys="urn:ietf:params:xml:ns:yang:ietf-ntp">
         <sys:associations>
         </sys:associations>
       </sys: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>20.1.1.1</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>
```

8.9. Global statistic

This example describes how to get clock current state -

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

9. 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.

10. 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 [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 inclide 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.

11. 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.

12. References

12.1. Normative 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>.

[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>.

12.2. Informative References

[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>.

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