On Implementing Time

draft-aanchal-time-implementation-guidance-00

Abstract

This document describes the properties of different types of time values available on digital systems and provides guidance on choices of these time values to the implementors of applications that use time in some form to provide the basic functionality and security guarantees.

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 May 3, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must
1. Introduction

The basic functionality and security guarantees claimed by many applications running on digital systems locally or in the Internet hinge on some notion of time. These applications have to choose one of the many types of time values available on the system, each of which has its own specific properties. However, currently these applications seem to be oblivious to the implications of choosing one or the other time value for implementation. This behaviour can be attributed to: a) the lack of clear understanding of the distinct properties of these time values, b) trade-offs of using one or the other for an application, and c) availability and compatibility of these time values on different operating systems.

In this document we describe the properties of various available time values on modern operating systems, discuss the trade-offs of using one over the other, and provide guidance to help implementors make an informed choice with some real-life examples.

2. Keeping Time: Different Clocks

Because time is relative to an observer, there cannot be a universally agreed upon time. At best we can achieve an approximation by updating our own observed time with a common reference time shared with other observers.

As this reference time is what we naively assume clocks on a wall are showing, we shall call it the "wall time." For most applications, it is based on the Universal Coordinated Time (UTC), an international standard time determined by averaging the output of several high-precision time-keeping devices. However, as UTC is following Earth’s solar time, it occasionally needs to be adjusted through leap seconds.

An individual computer system’s perception of time differs from this idealized wall time. Staying close to it requires some effort that comes with its own set of drawbacks. Systems therefore provide access to different types of clocks with different properties. Unfortunately, there is no standard terminology and definitions for these types. For the purpose of this document, we therefore define three different kinds of clocks that a system may or may not provide.
2.1. Raw Time

At its most fundamental, a system has its own perception of time; its unmodified, "raw time." This time is typically measured by counting cycles of an oscillator. Its quality therefore relies on the stability of this oscillator.

As it is a purely subjective time, no general meaning can be attached to any specific value. Only the amount of time passed can be determined by comparing two values.

Because raw time is unaltered, it is continuous and strictly monotonically increasing. Its value will always grow at a steady pace, never decrease, never make unexpected jumps, or stip. Such a time is sometimes called a "monotonic time."

2.2. Adjusted Raw Time

Even if highly accurate oscillators are used, raw time passes at a slightly different rate than wall time. This difference is called clock drift. It depends not only on the quality of the time source but also on environmental factors such as temperature.

When this drift is compensated by comparing the passage of raw time to some external time source that is considered to be closer to wall time, the result is "adjusted raw time." This adjustment doesn’t happen sporadically but rather, the rate of advance of time is slowed down or sped up slightly until it approaches the reference time again. As a result, adjusted raw time is still monotonic. Like raw time, adjusted raw time is subjective with no specific meaning attached to its values.

The most frequently used method of acquiring an external time source is through network timing protocols such as NTP [RFC5905]. As a result, adjusted raw time is susceptible to vulnerabilities of these protocols which may be exploited to maliciously manipulate this time.

2.3. Real Time

With adjusted raw time, a system already has access to a time that passes at a rate very similar to wall time. By adjusting the time value so that it represents the time passed since an epoch, a well-defined point of wall time such as seconds since midnight January 1st, 1970 on Unix systems, time values themselves gather meaning. The result is "real time."

While it is often assumed that real time is set to match wall time, this doesn’t need to be the case. A system’s operator is free to
change the value of real time at any time, likewise, system services such as a local NTP client may decide to do so.

As a consequence, real time is not monotonic. Not only may it jump forward, its value may even decrease.

2.4. Differences from Wall Time

These three clock types differ from wall time in three aspects:

- Both raw time and adjusted raw time can only represent differences in time by comparing two clock values. Only real time provides absolute time values that can be compared to wall time values.

- On the other hand, raw time and adjusted raw time are always monotonic whereas real time may experience sudden changes in value in either direction.

- Only adjusted raw time and real time are subject to external adjustments so that time passes at approximately the same rate as wall time. Raw time will over time drift away due to inevitable imperfections of the clock.

3. Expressing Time

Protocols or applications can express time in one of the two forms, depending on whether global agreement over the point in time is necessary.

3.1. Time Stamps

A "time stamp" expresses an absolute point in time. In order to reference the same point across multiple systems, it needs to be stated in wall time.

Time stamps are often used to express the validity of objects with a limited lifetime that are shared over the network. For instance, PKIX certificates [RFC5280] carry two time stamps expressing their earliest and latest validity.

In order to validate a time stamp, a system needs access to a clock that is reasonably close to wall time.

3.2. Time Spans

In contrast, a "time span" expresses a desired length of time. Examples of time spans are timeout values used in protocols to
determine packet loss or Time to Live (TTL) values that govern the lifetime of a local copy of an object.

While no access to wall time is necessary for correctly dealing with time spans, using a clock whose time passes at a different rate than wall time will result in different interpretations of time spans by different systems. However, in a network environment, the uncertainty introduced by differing transmission times is likely larger than that introduced by clock drift.

4. Current Implementations and Their Flaws

Currently, some software takes a common approach towards time stamps and time spans. Time stamps are registered with their wall time value, and time spans are registered with two time stamp values marking the start and the end of the span. Conversion of a time span into those time stamp markers is regularly based on real time.

Note that the start of a time span will be the current (real) time in case of a TTL. So, in case something needs to be cached for a certain time, the start time stamp is irrelevant and it is registered together with only the (real) expiration time.

Programmers might have had different reasons to base those markings on real time, for example:

1. A point in time is intuitively thought of as a wall clock time stamp. Time stamps from outside the software, which the software has to manage are already in wall clock time. The POSIX function to get the current (real) time which is regularly used for this, is gettimeofday(), which comes accross as something providing near wall clock time and which can be used for this purpose.

2. Managing time stamps and time span similarly, prevents code complexity.

For example, many software is organized around I/O event notification mechanisms like the POSIX select() and poll() system C API functions. These functions wait for a given time span for file descriptors to become ready to perform I/O. The given time span is determined by substracting the current real time value from smallest registered time stamp. When file descriptors are ready, the non-blocking I/O is performed, otherwise the given time span has passed and the action associated with the smallest registered time stamp needs to be performed.

For this programming pattern, a sorted list of time stamps has to be maintained by the software. To avoid coding complexity,
programmers might prefer a single list for both actual wall clock 
time stamps and those generated from real time to mark the end of 
a time span.

Using real time as a basis for the time stamps marking the start and 
end of a time span is bad because of the following reasons.

1. It can be set or overwritten manually,

2. It is subject to adjustments by timing protocols which on one 
   hand is important to make sure that this time is in sync with the 
   rest of the world but on the other hand makes it dependent on the 
   correctness and security of timing protocols.

Recent attacks [SECNTP], [MCBG] show how timing protocols like NTP 
can be leveraged to shift real time on systems.

Time stamps are always based on wall time, so the best one can do is 
to use real time while dealing with them. However, this limitation 
does not hold for the time spans. Managing time spans may be 
implemented in alternative ways which may prove to be more secure and 
robust.

An obvious question to ask is: Why do we need inception and 
expiration time stamps in the first place to define the validity 
period of cryptographic objects? Why can’t we just use time spans 
like TTL values instead? The reason is straightforward.

The authority determining and setting the validity period on the 
object can be different from the operator delivering the object. 
For example the TTL value on DNS resource records indicates to 
caching DNS resolvers how long to cache those records. These are 
an operational matter and are thus left to the operators of the 
DNS zone.

The content of the resource records are however determined by the 
signer of the records. When she is not also the zone operator, 
she has no way to determine when the records will be queried for, 
and thus has to depend on cryptographically signed wall clock 
based time stamps to limit the validity.

Note however that DNSSEC signatures do contain the original TTL of 
a resource record set, restricting the maximum TTL value with 
which the operator may deliver the resource records.
5. Alternative Approaches

For time spans, where we only need the rate of passage of time to be close enough to the rest of the world, one should not use the real time to establish the start and end time for the reasons mentioned above. The other two types of time are raw time and adjusted raw time. The important aspect of these monotonic time sources is not their current value but the guarantee that the time source is strictly linearly increasing and thus useful for calculating the difference in time between two samplings. But each comes with its own caveats.

Raw time is not subject to any adjustments by timing protocols, i.e., it is not adjusted for the error introduced by clock drift. This could have two repercussions. First, this makes correctness of raw time independent from the errors or security vulnerabilities of the timing protocols. Second, its correctness depends on the clock drift which further depends on various factors such as quality of the oscillator, work load, or ambient temperature on the system and may vary.

Adjusted raw time, on the other hand, is subject to adjustments by timing protocols. While it therefore compensates for the errors introduced by the drift of the local clock, this time can be incorrect as it is vulnerable to accuracy and security vulnerabilities of the underlying timing protocol.

The choice of time value to be used is application-specific. For instance in applications that can tolerate a certain amount of clock drift [CLOCKDRIFT], implementers can use raw time. However, if that is an issue then one has no choice but to fall back to adjusted raw time.

POSIX defines a system C API function which may provide raw time: clock_gettime(), when used with a clock_id of CLOCK_MONOTONIC (when supported by the system). POSIX does not make a distinction between raw time and adjusted raw time in the definition of this function. Beware that with some systems, CLOCK_MONOTONIC delivers adjusted raw time and that CLOCK_MONOTONIC_RAW needs to be used as clock_id to get unadjusted raw time. Non-POSIX systems may provide different APIs.

Software employing the pattern organized around I/O event notification mechanisms, as described in Section 4, should maintain two sorted lists of two different types of time stamps:

1. One to register events based on time stamps expressed in wall clock time
2. One to register the start and end of time spans in (adjusted) raw time

To determine the timeout value for a call to select() or poll(), the program needs to get the current time in both real time and in (adjusted) raw time. The current real time is subtracted from the lowest value of the time stamps expressed in wall time list. The current (adjusted) raw time from the lowest value of the time stamps expressed in (adjusted) raw time list. The lowest of the values should be used as the timeout value for select() or poll() and determines which action should be performed when the function times out.

Alternatively a single list of (adjusted) raw time could be used for both time stamps and time spans. In that case time stamps expressed in wall clock time should be converted into (adjusted) raw time, by first converting it into a time span by subtracting real time from it, and then adding the current time in (adjusted) raw time.

6. Acknowledgements

We are thankful to Sharon Goldberg and Benno Overreinder for useful discussions.

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

Time is a fundamental component for the security guarantees claimed by various applications. Therefore, any implementor concerned with security should be concerned with how these time values are implemented. This document discusses the security considerations with respect to implementing time values in applications in various sections.

9. Informative References


Authors’ Addresses

Aanchal Malhotra  
Boston University  
111 Cummington Mall  
Boston 02215  
USA  
Email: aanchal4@bu.edu

Martin Hoffmann  
Open Netlabs  
Science Park 400  
Amsterdam 1098 XH  
Netherlands  
Email: martin@opennetlabs.com

Willem Toorop  
NLnet Labs  
Science Park 400  
Amsterdam 1098 XH  
Netherlands  
Email: willem@nlnetlabs.nl
Network Time Protocol Best Current Practices
draft-ietf-ntp-bcp-07

Abstract

NTP Version 4 (NTPv4) has been widely used since its publication as RFC 5905 [RFC5905]. This documentation is a collection of Best Practices from across the NTP community.

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 February 1, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction ................................................. 3
   1.1. Requirements Language ................................. 3
2. Keeping NTP up to date ...................................... 3
   3.1. BCP 38 .............................................. 4
4. NTP Configuration Best Practices ............................. 4
   4.1. Use enough time sources ............................... 4
   4.2. Use a diversity of Reference Clocks ................... 5
   4.3. Control Messages ................................... 5
   4.4. Monitoring ......................................... 6
   4.5. Using Pool Servers .................................. 7
   4.6. Leap Second Handling ................................ 7
   4.6.1. Leap Smearing .................................. 8
5. NTP Security Mechanisms ................................... 9
   5.1. Pre-Shared Key Approach .............................. 9
   5.2. Autokey ............................................ 10
   5.3. Network Time Security ............................... 10
6. NTP Security Best Practices ................................ 10
   6.1. Minimizing Information Leakage ....................... 10
   6.2. Avoiding Daemon Restart Attacks ...................... 11
   6.3. Detection of Attacks Through Monitoring ............... 12
   6.4. KISS Packets ....................................... 13
   6.5. Broadcast Mode Should Only Be Used On Trusted Networks 13
   6.6. Symmetric Mode Should Only Be Used With Trusted Peers 14
7. NTP in Embedded Devices ................................... 15
   7.1. Updating Embedded Devices ............................ 15
   7.2. Server configuration ................................ 15
     7.2.1. Get a vendor subdomain for pool.ntp.org ........ 15
8. NTP over Anycast ............................................ 16
9. Acknowledgements .......................................... 17
10. IANA Considerations ..................................... 17
11. Security Considerations ................................ 17
12. References .............................................. 17
   12.1. Normative References ................................ 17
   12.2. Informative References ............................. 18
   12.3. URIs ............................................... 19
Appendix A. Implementation Specific Information .............. 20
   A.1. NTP Implementation by the Network Time Foundation 20
     A.1.1. Use enough time sources .......................... 20
     A.1.2. NTP Control and Facility Messages ............... 20
     A.1.3. Monitoring ....................................... 21
     A.1.4. Leap Second File ................................ 21
     A.1.5. Leap Smearing .................................. 22
1. Introduction

NTP Version 4 (NTPv4) has been widely used since its publication as RFC 5905 [RFC5905]. This documentation is a collection of Best Practices from across the NTP community.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Keeping NTP up to date

Many network security mechanisms rely on time as part of their operation. If attackers can spoof the time, they may be able to bypass or neutralize other security elements. For example, incorrect time can disrupt the ability to reconcile log file entries on the affected system with events on other systems. The best way to detect and protect computers and networks against undefined behavior and security threats related to time is to keep their NTP implementations current, use an appropriate number of trustworthy time sources, and properly monitor their time infrastructure.

There are always new ideas about security on the Internet, and an application which is secure today could be insecure tomorrow once an unknown bug (or a known behavior) is exploited in the right way. Even our definition of what is secure has evolved over the years, so code which was considered secure when it was written may turn out to be insecure after some time. By keeping NTP implementations current, having "enough" trustworthy time sources, and properly monitoring their time infrastructure, network operators can make sure that their time infrastructure is operating correctly and within specification, and is not being attacked or misused.

There are multiple versions of the NTP protocol in use, and multiple implementations in use, on many different platforms. It is recommended that that NTP users select an implementation that is actively maintained. Users should keep up to date on any known attacks on their selected implementation, and deploy updates containing security fixes as soon as practical.

3.1. BCP 38

Many network attacks rely on modifying the IP source address of a packet to point to a different IP address than the computer which originated it. UDP-based protocols such as NTP are generally more susceptible to spoofing attacks than other connection-oriented protocols. NTP control messages can generate a lot of data in response to a small query, which makes it more attractive as a vector for distributed denial-of-service attacks. (NTP Control messages are discussed further in Section 4.3). Mitigating source address spoofing attacks should be a priority of anyone administering NTP.

BCP 38 [RFC2827] was approved in 2000 to address this. BCP 38 [RFC2827] calls for filtering outgoing and incoming traffic to make sure that the source and destination IP addresses are consistent with the expected flow of traffic on each network interface. It is recommended that all networks (and ISP’s of any size) implement ingress and egress filtering. More information is available at the BCP38 Info page [1].

4. NTP Configuration Best Practices

This section provides general Best Practices. Best Practices that are implementation specific are compiled in Section Appendix A.

4.1. Use enough time sources

An NTP implementation (as opposed to an SNTP implementation) takes the available sources of time and submits this timing data to sophisticated intersection, clustering, and combing algorithms to get the best estimate of the correct time. The description of these algorithms is beyond the scope of this document. Interested readers should read RFC 5905 [RFC5905] or the detailed description of NTP in MILLS 2006 [MILLS2006]

- If there is only 1 source of time, the answer is obvious. It may not be a good source of time, but it’s the only source of time that can be considered. Any issue with the time at the source will be passed on to the client.

- If there are 2 sources of time and they agree well enough, then the best "time" can be calculated easily. But if one source fails, then the solution degrades to the single-source solution outlined above. And if the two sources don’t agree, then it’s impossible to know which one is correct by simply looking at the time.
o If there are 3 sources of time, there is more data available to converge on a "best" time, and this time is more likely to be accurate. And the loss of one of the sources (by becoming unreachable or unusable) can be tolerated. But at that point, the solution degrades to the 2 source solution.

o 4 or more sources of time is better. If one of these sources develops a problem there are still at least 3 other time sources.

But even with 4 or more sources of time, systemic problems can happen. During the leap second of June of 2015, several operators implemented leap smearing while others did not, and many NTP end nodes could not determine an accurate time source because 2 of their 4 sources of time gave them consistent UTC/POSIX time, while the other 2 gave them consistent leap-smeared time. See Section 4.6.1 for more information.

Monitor your NTP instances. If your time sources do not generally agree, find out why and either correct the problems or stop using defective servers. See Section 4.4 for more information.

4.2. Use a diversity of Reference Clocks

When using servers with attached hardware reference clocks, it is recommended that several different types of reference clocks be used. Having a diversity of sources means that any one issue is less likely to cause a service interruption.

Are all clocks on a network from the same vendor? They may have the same bugs. Are they using the same base chipset, regardless of whether or not the finished products are from different vendors? Are they all running the same version of firmware? chipset and firmware bugs can happen, but they can be more difficult to diagnose than application software bugs.

A systemic problem with time from any satellite navigation service is possible and has happened. Sunspot activity can render satellite or radio-based time source unusable. If the time on your network must be correct close to 100% of the time, then even if you are using a satellite-based system, you must plan for those rare instances when the system is unavailable (or wrong!).

4.3. Control Messages

Some implementations of NTPv4 provide the NTP Control Messages which originally have been specified in Appendix B of [RFC1305] which defined NTPv3, but never have been part of the NTPv4 specification.
The NTP Control Messages are designed to permit monitoring and optionally authenticated control of NTP and its configuration. Used properly, these facilities provide vital debugging and performance information and control. Used improperly, these facilities can be an abuse vector.

The ability to use Mode 6 beyond its basic monitoring capabilities can be limited to authenticated sessions that provide a ‘controlkey’.

The NTP Control Messages responses are much larger than the corresponding queries. Thus, they can be abused in high-bandwidth DDoS attacks. To provide protection for such abuse NTP server operators should deploy ingress filtering BCP 38 [RFC2827].

4.4. Monitoring

Use your NTP implementation’s remote monitoring capabilities to quickly identify servers which are out of sync, and ensure correctness of the service. Monitor system logs for messages so problems and abuse attempts can be quickly identified.

If a system starts getting unexpected time replies from its time servers, that can be an indication that the IP address of the system is being forged in requests to its time server, and these abusers are trying to convince that time server to stop serving time to that system.

If a system is a broadcast client and its syslog shows that it is receiving "early" time messages from its server, that is an indication that somebody may be forging packets from a broadcast server.

If a server’s syslog shows messages that indicates it is receiving timestamps that are earlier than the current system time, then either the system clock is unusually fast or somebody is trying to launch a replay attack against that server.

If a system is using broadcast mode and is running ntp-4.2.8p6 or later, use the 4th field of the ntp.keys file to specify the IPs of machines that are allowed to serve time to the group.
4.5. Using Pool Servers

It only takes a small amount of bandwidth and system resources to synchronize one NTP client, but NTP servers that can service tens of thousands of clients take more resources to run. Users who want to synchronize their computers should only synchronize to servers that they have permission to use.

The NTP pool project is a group of volunteers who have donated their computing and bandwidth resources to freely distribute time from primary time sources to others on the Internet. The time is generally of good quality, but comes with no guarantee whatsoever. If you are interested in using the pool, please review their instructions at http://www.pool.ntp.org/en/use.html [2].

If you are a vendor who wishes to provide time service to your customers or clients, consider joining the pool and providing a "vendor zone" through the pool project.

If you want to synchronize many computers, consider running your own NTP servers that are synchronized by the pool, and synchronizing your clients to your in-house NTP servers. This reduces the load on the pool.

If you would like to contribute a server with a static IP address and a permanent Internet connection to the pool, please consult the instructions at http://www.pool.ntp.org/en/join.html [3].

4.6. Leap Second Handling

UTC is kept in agreement with the astronomical time UT1 [4] to within +/- 0.9 seconds by the insertion (or possibly a deletion) of a leap second. UTC is an atomic time scale whereas UT1 is based on the rotational rate of the earth. Leap seconds are not introduced at a fixed rate. They are announced by the IERS (International Earth Rotation and Reference Systems Service) in its Bulletin C [5] when necessary to keep UTC and UT1 aligned.

NTP time is based on the UTC timescale, and the protocol has the capability to broadcast leap second information. Some GNSS systems (like GPS) or radio transmitters (like DCF77) broadcast leap second information, so if you are synced to an ntp server that is ultimately synced to a source that provides leap second notification you will get advance notification of impending leap seconds automatically.

Since the length of the UT1 day is generally slowly increasing [6], all leap seconds that have been introduced since the practice started in 1972 have been "positive" leap seconds, where a second is added to
UTC. NTP also supports a "negative" leap second, where a second is removed from UTC, should that ever become necessary.

While earlier versions of NTP contained some ambiguity regarding when a leap second that is broadcast by a server should be applied by a client, RFC 5905 is clear that leap seconds are only applied on the last day of a month. However, because some older clients may apply it at the end of the current day, it is recommended that NTP servers wait until the last day of the month before broadcasting leap seconds. Doing this will prevent older clients from applying a leap second at the wrong time. Note well that NTPv4 allows a maximum poll interval of 17, or 131,072 seconds, which is longer than a day.

The IETF maintains a leap second list [?] for NTP users who are not receiving leap second information through an automatic source.

Files are also available from other sources:

NIST: ftp://time.nist.gov/pub/leap-seconds.list


IERS (announces leap seconds):
https://hpiers.obspm.fr/iers/bul/bulc/ntp/leap-seconds.list

See Appendix A.1.4 for instructions on applying the leap second file to the reference implementation.

4.6.1. Leap Smearing

Some NTP installations may instead make use of a technique called "Leap Smearing". With this method, instead of introducing an extra second (or eliminating a second), NTP time will be slewed in small increments over a comparably large window of time (called the smear interval) around the leap second event. The smear interval should be large enough to make the rate that the time is slewed small, so that clients will follow the smeared time without objecting. Periods ranging from 2 to 24 hours have been used successfully. During the adjustment window, all the NTP clients' times may be offset from UTC by as much as a full second, depending on the implementation. But at least all clients will generally agree on what time they think it is!

NOTE WELL that using a leap-smear can cause your reported time to be "legally indefensible" and/or be a breach of compliance regulations.

The purpose of Leap Smearing is to enable systems that don't deal with the leap second event properly to function consistently, at the
expense of fidelity to UTC during the smear window. During a standard leap second event, that minute will have 61 (or possibly 59) seconds in it, and some applications (and even some OS's) are known to have problems with that.

Clients that are connected to leap smearing servers MUST NOT apply the "standard" NTP leap second handling. So if they are using ntpd, these clients must never have a leap second file loaded, and the smearing servers must never advertise to clients that a leap second is pending.

Leap Smearing MUST NOT be used for public-facing NTP servers, as they will disagree with non-smearing servers (as well as UTC) during the leap smear interval. However, be aware that some public-facing servers may be configured this way anyway in spite of this guidance.

System Administrators are advised to be aware of impending leap seconds and how the servers (inside and outside their organization) they are using deal with them. Individual clients must never be configured to use a mixture of smeared and non-smeared servers. If a client uses smeared servers, the servers it uses must all have the same leap smear configuration.

5. NTP Security Mechanisms

In the standard configuration NTP packets are exchanged unprotected between client and server. An adversary that is able to become a Man-In-The-Middle is therefore able to drop, replay or modify the content of the NTP packet, which leads to degradation of the time synchronization or the transmission of false time information. A profound threat analysis for time synchronization protocols are given in RFC 7384 [RFC7384]. NTP provides two internal security mechanisms to protect authenticity and integrity of the NTP packets. Both measures protect the NTP packet by means of a Message Authentication Code (MAC). Neither of them encrypts the NTP’s payload, because this payload information is not considered to be confidential.

5.1. Pre-Shared Key Approach

This approach applies a symmetric key for the calculation of the MAC, which protects authenticity and integrity of the exchanged packets for an association. NTP does not provide a mechanism for the exchange of the keys between the associated nodes. Therefore, for each association, keys have to be exchanged securely by external means. It is recommended that each association be protected by its own unique key. NTP does not provide a mechanism to automatically refresh the applied keys. It is therefore recommended that the participants periodically agree on a fresh key. The calculation of
the MAC may always be based on an MD5 hash, and an AES-128-CMAC hash
is expected to soon be allowed as well. If the NTP daemon is built
against an OpenSSL library, NTP can also base the calculation of the
MAC upon any other digest algorithm supported by each side’s OpenSSL
library.

To use this approach the communication partners have to exchange the
key, which consists of a keyid with a value between 1 and 65534,
inclusive, and a label which indicates the chosen digest algorithm.
Each communication partner adds this information to its own key file.

Some implementations store the key in clear text. Therefore it
should only be readable by the NTP process. Different keys are added
line by line to the key file.

An NTP client establishes a protected association by appending the
key to the server statement in its configuration file. Note that the
NTP process has to trust the applied key.

5.2. Autokey

Autokey was designed in 2003 to provide a means for clients to
authenticate servers. However, security researchers have identified
vulnerabilities in the Autokey protocol, which make the protocol
"useless". [8]

Autokey SHOULD NOT BE USED.

5.3. Network Time Security

Work is in progress on an enhanced replacement for Autokey, which is
called Network Time Security (NTS) [NTSFORNTP]. As of July 2018,
this effort was at draft #12, and in the ’Working Group Last Call’
process. Readers are encouraged to adopt its mechanisms.

6. NTP Security Best Practices

This section lists some general NTP security practices, but these
concepts may (or may not) have been mitigated in particular versions
of particular implementations. Contact the maintainers of your
implementation for more information.

6.1. Minimizing Information Leakage

The base NTP packet leaks important information (including reference
ID and reference time) that may be used in attacks [NDSS16],
[CVE-2015-8138], [CVE-2016-1548]. A remote attacker can learn this
information by sending mode 3 queries to a target system and
inspecting the fields in the mode 4 response packet. NTP control queries also leak important information (including reference ID, expected origin timestamp, etc.) that may be used in attacks [CVE-2015-8139]. A remote attacker can learn this information by sending control queries to a target system and inspecting the response.

As such, access control should be used to limit the exposure of this information to inappropriate third parties.

Hosts should only respond to NTP control queries from authorized parties. One way to do this is to only allow control queries from authenticated sources via authorized IP addresses.

A host that is not supposed to act as an NTP server that provides timing information to other hosts may additionally log and drop incoming mode 3 timing queries from unexpected sources. Note well that the easiest way to monitor ntpd’s status is to send it a mode 3 query. A much better approach might be to filter mode 3 queries at the edge, or make sure mode 3 queries are allowed only from trusted systems or networks.

A "leaf-node host" is a host that is using NTP solely for the purpose of adjusting its own system time. Such a host is not expected to provide time to other hosts, and relies exclusively on NTP’s basic mode to take time from a set of servers. (That is, the host sends mode 3 queries to its servers and receives mode 4 responses from these servers containing timing information.) To minimize information leakage, leaf-node hosts should drop all incoming NTP packets except mode 4 response packets that come from known sources. Note well that proper monitoring of an ntpd instance includes checking the time of that ntpd instance.

6.2. Avoiding Daemon Restart Attacks

RFC 5905 [RFC5905] says NTP clients should not accept time shifts greater than the panic threshold. Specifically, RFC 5905 says "PANIC means the offset is greater than the panic threshold PANICT (1000 s) and SHOULD cause the program to exit with a diagnostic message to the system log."

However, this behavior can be exploited by attackers [NDSS16], when the following two conditions hold:

1. The operating system automatically restarts the NTP daemon when it quits. (Modern *NIX operating systems are replacing traditional init systems with process supervisors, such as systemd, which can be configured to automatically restart any
daemons that quit. This behavior is the default in CoreOS and Arch Linux. It is likely to become the default behavior in other systems as they migrate legacy init scripts to process supervisors such as systemd.)

2. If, against long-standing recommendation, ntpd is always started with the -g option, it will ignore the panic threshold when it is restarted. The -g option SHOULD only be provided in cold-start situations.

In such cases, if the attacker can send the target an offset that exceeds the panic threshold, the client will quit. Then, when the client restarts, it ignores the panic threshold and accepts the attacker’s large offset.

Hosts running with the above two conditions should be aware that the panic threshold does not protect them from attacks. The recommended and natural solution is not to run hosts with these conditions. Specifically, only ignore the panic threshold in cold-start situations if sufficient oversight and checking is in place to make sure that this is appropriate.

As an alternative, the following steps could be taken to mitigate the risk of attack.

- Monitor NTP system log to detect when the NTP daemon has quit due to a panic event, as this could be a sign of an attack.
- Request manual intervention when a timestep larger than the panic threshold is detected.
- Prevent the NTP daemon from taking time steps that set the clock to a time earlier than the compile date of the NTP daemon.
- Add "minsane" and "minclock" parameters to the ntp.conf file so ntpd waits until "enough" trusted sources of time agree on the correct time.

6.3. Detection of Attacks Through Monitoring

Users should monitor their NTP instances to detect attacks. Many known attacks on NTP have particular signatures. Common attack signatures include:

1. "Bogus packets" - A packet whose origin timestamp does not match the value that expected by the client.

3. A packet with an invalid cryptographic MAC [CCR16].

The observation of many such packets could indicate that the client is under attack.

Also, Kiss-o’-Death (KoD) packets can be used in denial of service attacks. Thus, the observation of even just one KoD packet with a high poll value could be sign that the client is under attack. See Section 6.4 for more information.

6.4. KISS Packets

The "Kiss-o’-Death" (KoD) packet is a rate limiting mechanism where a server can tell a misbehaving client to "back off" its query rate. It is important for all NTP devices to respect these packets and back off when asked to do so by a server. It is even more important for an embedded device, which may not have exposed a control interface for NTP.

That said, a client must only accept a KoD packet if it has a valid origin timestamp. Once a RATE packet is accepted, the client should increase its poll interval value (thus decreasing its polling rate) up to a reasonable maximum. This maximum can vary by implementation but should not exceed a poll interval value of 13 (2 hours). The mechanism to determine how much to increase the poll interval value is undefined in RFC 5905 [RFC5905]. If the client uses the poll interval value sent by the server in the KoD packet, it must not simply accept any value. Using large interval values may open a vector for a denial-of-service attack that causes the client to stop querying its server [NDSS16].

The KoD mechanism relies on clients behaving properly in order to be effective. Some clients ignore the KoD packet entirely, and other poorly-implemented clients might unintentionally increase their poll rate and simulate a denial of service attack. Server administrators should be prepared for this and take measures outside of the NTP protocol to drop packets from misbehaving clients.

6.5. Broadcast Mode Should Only Be Used On Trusted Networks

Per RFC 5905 [RFC5905], NTP’s broadcast mode is authenticated using symmetric key cryptography. The broadcast server and all of its broadcast clients share a symmetric cryptographic key, and the broadcast server uses this key to append a message authentication code (MAC) to the broadcast packets it sends.
Importantly, all broadcast clients that listen to this server must know the cryptographic key. This means that any client can use this key to send valid broadcast messages that look like they come from the broadcast server. Thus, a rogue broadcast client can use its knowledge of this key to attack the other broadcast clients.

For this reason, an NTP broadcast server and all its client must trust each other. Broadcast mode should only be run from within a trusted network.

Starting with ntp-4.2.8p7 the ntp.keys file accepts an optional 4th column, a comma-separated list of IPs that are allowed to serve time. Use this feature. Note, however, that an adversarial client that knows the symmetric broadcast key could still easily spoof its source IP to an IP that is allowed to serve time. (This is easy to do because the origin timestamp on broadcast mode packets is not validated by the client. By contrast, client/server and symmetric modes do require origin timestamp validation, making it more difficult to spoof packets [CCR16].)

6.6. Symmetric Mode Should Only Be Used With Trusted Peers

In symmetric mode, two peers Alice and Bob can both push and pull synchronization to and from each other using either ephemeral symmetric passive (mode 2) or persistent symmetric active (NTP mode 1) packets. The persistent association is preconfigured and initiated at the active peer but not preconfigured at the passive peer (Bob). Upon receipt of a mode 1 NTP packet from Alice, Bob mobilizes a new ephemeral association if he does not have one already. This is a security risk for Bob because an arbitrary attacker can attempt to change Bob’s time by asking Bob to become its symmetric passive peer.

For this reason, a host (Bob) should only allow symmetric passive associations to be established with trusted peers. Specifically, Bob should require each of its symmetric passive association to be cryptographically authenticated. Each symmetric passive association should be authenticated under a different cryptographic key.

The use of a different cryptographic key per peer prevents a Sybil attack, where a single malicious peer uses the same cryptographic key to set up multiple symmetric associations a target, and thus bias the results of the target’s Byzantine fault tolerant peer selection algorithms.

Starting with ntp-4.2.8p7 the ntp.keys file accepts an optional 4th column, a comma-separated list of IPs that are allowed to serve time. Use this feature.
7. NTP in Embedded Devices

Readers of this BCP already understand how important accurate time is for network computing. And as computing becomes more ubiquitous, there will be many small "Internet of Things" devices that require accurate time. These embedded devices may not have a traditional user interface, but if they connect to the Internet they will be subject to the same security threats as traditional deployments.

7.1. Updating Embedded Devices

Vendors of embedded devices have a special responsibility to pay attention to the current state of NTP bugs and security issues, because their customers don’t have the ability to update their NTP implementation on their own. Those devices may have a single firmware upgrade, provided by the manufacturer, that updates all capabilities at once. This means that the vendor assumes the responsibility of making sure their devices have the latest NTP updates applied.

This should also include the ability to update any NTP server addresses on these devices.

There is a catalog of NTP server abuse incidents, some of which involve embedded devices, on the Wikipedia page for NTP Server Misuse and Abuse [9].

7.2. Server configuration

Vendors of embedded devices that need time synchronization should also carefully consider where they get their time from. There are several public-facing NTP servers available, but they may not be prepared to service requests from thousands of new devices on the Internet.

Vendors are encouraged to invest resources into providing their own time servers for their devices to connect to.

7.2.1. Get a vendor subdomain for pool.ntp.org

The NTP Pool Project offers a program where vendors can obtain their own subdomain that is part of the NTP Pool. This offers vendors the ability to safely make use of the time distributed by the Pool for their devices. Vendors are encouraged to support the pool if they participate. For more information, visit http://www.pool.ntp.org/en/vendors.html [10].
8. NTP over Anycast

Anycast is described in BCP 126 [RFC4786]. (Also see RFC 7094 [RFC7094]). With anycast, a single IP address is assigned to multiple interfaces, and routers direct packets to the closest active interface.

Anycast is often used for Internet services at known IP addresses, such as DNS. Anycast can also be used in large organizations to simplify configuration of a large number of NTP clients. Each client can be configured with the same NTP server IP address, and a pool of anycast servers can be deployed to service those requests. New servers can be added to or taken from the pool, and other than a temporary loss of service while a server is taken down, these additions can be transparent to the clients.

NOTE WELL: Using a single anycast address for NTP should be done with care. It means each client will likely use a single time server source. A key element of a robust NTP deployment is each client using multiple sources of time. With multiple time sources, a client will analyze the various time sources, selecting good ones, and disregarding poor ones. If a single Anycast address is used, this analysis will not happen.

If clients are connected to an NTP server via anycast, the client does not know which particular server they are connected to. As anycast servers may arbitrarily enter and leave the network, the server a particular client is connected to may change. This may cause a small shift in time from the perspective of the client when the server it is connected to changes. It is recommended that anycast only be deployed in environments where these small shifts can be tolerated.

Configuration of an anycast interface is independent of NTP. Clients will always connect to the closest server, even if that server is having NTP issues. It is recommended that anycast NTP implementations have an independent method of monitoring the performance of NTP on a server. If the server is not performing to specification, it should remove itself from the Anycast network. It is also recommended that each Anycast NTP server have at least one Unicast interface, so its performance can be checked independently of the anycast routing scheme.

One useful application in large networks is to use a hybrid unicast/anycast approach. Stratum 1 NTP servers can be deployed with unicast interfaces at several sites. Each site may have several Stratum 2 servers with two ethernet interfaces. One interface has a unique unicast IP address. The second has an anycast IP interface (with a
shared IP address per location). The unicast interfaces can be used to obtain time from the Stratum 1 servers globally (and perhaps peer with the other Stratum 2 servers at their site). Clients at each site can be configured to use the shared anycast address for their site, simplifying their configuration. Keeping the anycast routing restricted on a per-site basis will minimize the disruption at the client if its closest anycast server changes. Each Stratum 2 server can be uniquely identified on their unicast interface, to make monitoring easier.

9. Acknowledgements

The authors wish to acknowledge the contributions of Sue Graves, Samuel Weiler, Lisa Perdue, Karen O’Donoghue, David Malone, Sharon Goldberg, Martin Burnicki, Miroslav Lichvar, Daniel Fox Franke, and Robert Nagy.

10. IANA Considerations

This memo includes no request to IANA.

11. Security Considerations

Time is a fundamental component of security on the internet. Credentials and certificates can expire. Logins and other forms of access can be revoked after a period of time, or at a scheduled time. And some applications may assume that system time cannot be changed and is always monotonic, and vulnerabilities may be exposed if a time in the past is forced into a system. Therefore, any system administrator concerned with security should be concerned with how the current time gets into their system.

[NTSFORNTP] is an Internet-Draft that specifies the Network Time Security (NTS) mechanism and applies it specifically to NTP. Readers are encouraged to check the status of the draft, and make use of the methods it describes.

12. References

12.1. Normative References


12.2. Informative References


12.3. URIs


[13] https://support.ntp.org/bin/view/Support/ConfiguringNTP
Appendix A. Implementation Specific Information

This appendix provides information that is specific to various implementation of RFC 5905.

A.1. NTP Implementation by the Network Time Foundation

The Network Time Foundation (NTF) provides the reference implementation of NTP, well-known under the name "ntpd". It is actively maintained and developed by NTF’s NTP Project, with help from volunteers and NTF’s supporters. This NTP software can be downloaded from ntp.org [11].

A.1.1. Use enough time sources

In addition to the recommendation given in Section 4.1 the ntpd implementation provides the 'pool' directive. Starting with ntp-4.2.6, this directive will spin up "enough" associations to provide robust time service, and will disconnect poor servers and add in new servers as-needed. If you have good reason, you may use the 'minclock' and 'maxclock' options of the 'tos' command to override the default values of how many servers are discovered through the 'pool' directive.

A.1.2. NTP Control and Facility Messages

In addition to NTP Control Messages the ntpd implementation also offers the mode 7 commands for monitoring and configuration.

If Mode 7 has been explicitly enabled to be used for more than basic monitoring it should be limited to authenticated sessions that provide a 'requestkey'.

As mentioned above, there are two general ways to use Mode 6 and Mode 7 requests. One way is to query ntpd for information, and this mode can be disabled with:

restrict ... noquery

The second way to use Mode 6 and Mode 7 requests is to modify ntpd’s behavior. Modification of ntpd’s configuration requires an authenticated session BY default. If no authentication keys have been specified no modifications can be made. For additional protection, the ability to perform these modifications can be controlled with:

restrict ... nomodify
Users can prevent their NTP servers from considering query/configuration traffic by default by adding the following to their ntp.conf file:

```plaintext
restrict default -4 nomodify notrap nopeer noquery
restrict default -6 nomodify notrap nopeer noquery
restrict source nomodify notrap noquery
# nopeer is OK if you don’t use the ‘pool’ directive
```

A.1.3. Monitoring

The reference implementation of NTP allows remote monitoring. Access to this service is generally controlled by the "noquery" directive in NTP’s configuration file (ntp.conf) via a "restrict" statement. The syntax reads:

```plaintext
restrict address mask address_mask noquery
```

If a system is using broadcast mode and is running ntp-4.2.8p6 or later, use the 4th field of the ntp.keys file to specify the IPs of machines that are allowed to serve time to the group.

A.1.4. Leap Second File

The use of leap second files requires ntpd 4.2.6 or later. After fetching the leap seconds file onto the server, add this line to ntpd.conf to apply and use the file:

```plaintext
leapfile "/path/to your/leap-file"
```

You may need to restart ntpd to apply this change.

ntp servers with a manually configured leap second file will ignore leap second information broadcast from upstream NTP servers until the leap second file expires. If no valid leap second file is available then a leap second notification from an attached reference clock is always accepted by ntpd.

If no valid leap second file is available, a leap second notification may be accepted from upstream NTP servers. As of ntp-4.2.6, a majority of servers must provide the notification before it is accepted. Before 4.2.6, a leap second notification would be accepted if a single upstream server of a group of configured servers provided a leap second notification. This would lead to misbehavior if single NTP servers sent an invalid leap second warning, e.g. due to a faulty GPS receiver in one server, but this behavior was once chosen because
in the "early days" there was a greater chance that leap second
information would be available from a very limited number of sources.

A.1.5. Leap Smearing

Leap Smearing was introduced in ntpd versions 4.2.8.p3 and 4.3.47, in
response to CLIENT requests. Support for leap smearing is not
configured by default and must be added at compile time. In
addition, no leap smearing will occur unless a leap smear interval is
specified in ntpd.conf. For more information, refer to

A.1.6. Configuring ntpd

See https://support.ntp.org/bin/view/Support/ConfiguringNTP [13] for
additional information on configuring ntpd.

A.1.7. Pre-Shared Keys

Each communication partner must add the keyid information to their
key file in the form:

keyid label key

An ntpd client establishes a protected association by appending the
option "key keyid" to the server statement in ntp.conf:

server address key keyid

A key is deemed trusted when its keyid is added to the list of
trusted keys by the "trustedkey" statement in ntp.conf.

trustedkey keyid_1 keyid_2 ... keyid_n

Authors' Addresses

Denis Reilly (editor)
Spectracom
1565 Jefferson Road, Suite 460
Rochester, NY  14623
US

Email: denis.reilly@spectracom.orolia.com
Harlan Stenn  
Network Time Foundation  
P.O. Box 918  
Talent, OR 97540  
US  
Email: stenn@nwtime.org

Dieter Sibold  
Physikalisch-Technische Bundesanstalt  
Bundesallee 100  
Braunschweig D-38116  
Germany  
Phone: +49-(0)531-592-8420  
Fax: +49-531-592-698420  
Email: dieter.sibold@ptb.de
NTP Client Data Minimization
draft-ietf-ntp-data-minimization-02

Abstract

This memo proposes backward-compatible updates to the Network Time Protocol to strip unnecessary identifying information from client requests and to improve resilience against blind spoofing of unauthenticated server responses.

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 1, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Network Time Protocol (NTP) packets, as specified by RFC 5905 [RFC5905], carry a great deal of information about the state of the NTP daemon which transmitted them. In the case of mode 4 packets (responses sent from server to client), as well as in broadcast (mode 5) and symmetric peering modes (mode 1/2), most of this information is essential for accurate and reliable time synchronization. However, in mode 3 packets (requests sent from client to server), most of these fields serve no purpose. Server implementations never need to inspect them, and they can achieve nothing by doing so. Populating these fields with accurate information is harmful to privacy of clients because it allows a passive observer to fingerprint clients and track them as they move across networks.

This memo updates RFC 5905 to redact unnecessary data from mode 3 packets. This is a fully backwards-compatible proposal. It calls for no changes on the server side, and clients which implement these updates will remain fully interoperable with existing servers.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Client Packet Format

In every client-mode packet sent by a Network Time Protocol [RFC5905] implementation:
The first octet, which contains the leap indicator, version number, and mode fields, SHOULD be set to 0x23 (LI = 0, VN = 4, Mode = 3).

The Transmit Timestamp field SHOULD be set uniformly at random, generated by a mechanism suitable for cryptographic purposes. [RFC4086] provides guidance on the generation of random values.

The Poll field MAY be set to the actual polling interval as specified by RFC 5905, or else MAY be set to zero.

The Precision field SHOULD be set to 0x20.

All other header fields, specifically the Stratum, Root Delay, Root Dispersion, Reference ID, Reference Timestamp, Origin Timestamp, and Receive Timestamp, SHOULD be set to zero.

Servers MUST allow client packets to conform to the above recommendations. This requirement shall not be construed so as to prohibit servers from rejecting conforming packets for unrelated reasons, such as access control or rate limiting.

4. Security and Privacy Considerations

4.1. Data Minimization

Zeroing out unused fields in client requests prevents disclosure of information that can be used for fingerprinting [RFC6973].

While populating any of these fields with authentic data reveals at least some identifying information about the client, the Origin Timestamp and Receive Timestamp fields constitute a particularly severe information leak. RFC 5905 calls for clients to copy the transmit timestamp and destination timestamp of the server’s most recent response into the origin timestamp and receive timestamp (respectively) of their next request to that server. Therefore, when a client moves between networks, a passive observer of both network paths can determine with high confidence that the old and new IP addresses belong to the same system by noticing that the transmit timestamp of a response sent to the old IP matches the origin timestamp of a request sent from the new one.

Zeroing the poll field is made optional (MAY rather than SHOULD) so as not to preclude future development of schemes wherein the server uses information about the client’s current poll interval in order to recommend adjustments back to the client. Putting accurate information into this field has no significant impact on privacy.
since an observer can already obtain this information simply by observing the actual interval between requests.

4.2. Transmit Timestamp Randomization

While this memo calls for most fields in client packets to be set to zero, the transmit timestamp is randomized. This decision is motivated by security as well as privacy.

NTP servers copy the transmit timestamp from the client’s request into the origin timestamp of the response; this memo calls for no change in this behavior. Clients discard any response whose origin timestamp does not match the transmit timestamp of any request currently in flight.

In the absence of cryptographic authentication, verification of origin timestamps is clients’ primary defense against blind spoofing of NTP responses. It is therefore important that clients’ transmit timestamps be unpredictable. Their role in this regard is closely analogous to that of TCP Initial Sequence Numbers [RFC6528].

The traditional behavior of the NTP reference implementation is to randomize only a few (typically 10-15 depending on the precision of the system clock) low-order bits of transmit timestamp, with all higher bits representing the system time, as measured just before the packet was sent. This is suboptimal, because with so few random bits, an adversary sending spoofed packets at high volume will have a good chance of correctly guessing a valid origin timestamp.

5. IANA Considerations

[RFC EDITOR: DELETE PRIOR TO PUBLICATION]

This memo introduces no new IANA considerations.

6. Implementation status - RFC EDITOR: REMOVE BEFORE PUBLICATION

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in 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.

As of today the following vendors have produced an implementation of the NTP Client Data Minimization recommendations described in this document.

OpenNTPD

7. References

7.1. Normative References


7.2. Informative References


7.3. URIs

Appendix A. Acknowledgements

The authors would like to gratefully acknowledge Henning Brauer for pioneering NTP data minimization techniques as early as June 2004 [1] as part of an NTP implementation for the OpenBSD Project.

The authors would like to thank Prof. Sharon Goldberg and Miroslav Lichvar for encouraging standardisation of the approach described in this document.

Authors’ Addresses

Daniel Fox Franke
Akamai Technologies, Inc.
150 Broadway
Cambridge, MA  02142
United States

Email: dafranke@akamai.com
URI:   https://www.dfranke.us

Aanchal Malhotra
Boston University
111 Cummington St
Boston, MA/  02215
United States

Email: aanchal4@bu.edu
Abstract

RFC 5905 [RFC5905] states that Network Time Protocol (NTP) packets should be authenticated by appending a 128-bit key to the NTP data, and hashing the result with MD5 to obtain a 128-bit tag. This document deprecates MD5-based authentication, which is considered to be too weak, and recommends the use of AES-CMAC [RFC4493] as a replacement.

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 September 6, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

Malhotra & Goldberg Expires September 6, 2018 [Page 1]
1. Introduction

RFC 5905 [RFC5905] states that Network Time Protocol (NTP) packets should be authenticated by appending a 128-bit key to the NTP data, and hashing the result with MD5 to obtain a 128-bit tag. This document deprecates MD5-based authentication, which is considered to be too weak, and recommends the use of AES-CMAC [RFC4493] as a replacement.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Deprecating MD5

RFC 5905 [RFC5905] defines how the MD5 digest algorithm in RFC 1321 [RFC1321] can be used as a message authentication code (MAC) for authenticating NTP packets. However, as discussed in [BCK] and RFC 6151 [RFC6151], this is not a secure MAC and therefore MUST be deprecated.

3. Replacement Recommendation

If authentication is implemented, then AES-CMAC as specified in RFC 4493 [RFC4493] SHOULD be computed over all fields in the NTP header, and any extension fields that are present in the NTP packet as described in RFC 5905 [RFC5905]. The MAC key for NTP MUST be at

Malhotra & Goldberg Expires September 6, 2018 [Page 2]
least 128 bits long AES-128 key and the resulting MAC tag MUST be at least 128 bits long as stated in section 2.4 of RFC 4493 [RFC4493]. NTP makes this transition possible as it supports algorithm agility as described in Section 2.1 of RFC 7696 [RFC7696].

The hosts who wish to use NTP authentication share a symmetric key out-of-band. So they MUST implement AES-CMAC and share the corresponding symmetric key. A symmetric key is a triplet of ID, type (e.g. MD5, AES-CMAC) and the key itself. All three have to match in order to successfully authenticate packets between two hosts. Old implementations that don’t support AES-CMAC will not accept and will not send packets authenticated with such a key.

4. Motivation

AES-CMAC is recommended for the following reasons:

1. It is an IETF standard that is available in many open source implementations.

2. It is immune to nonce-reuse vulnerabilities (e.g. [Joux]) because it does not use a nonce.

3. It has fine performance in terms of latency and throughput.

4. It benefits from native hardware support, for instance, Intel’s New Instruction set.

5. Test Vectors

For test vectors and their outputs refer to Section 4 of RFC 4493 [RFC4493]

6. Security Considerations

Refer to the Appendices A, B and C of NIST document [NIST] and Security Considerations Section of RFC 4493 [RFC4493] for discussion on security guarantees of AES-CMAC.

7. Acknowledgements

The authors wish to acknowledge useful discussions with Leen Alshenibr, Daniel Franke, Ethan Heilman, Kenny Paterson, Leonid Reyzin, Harlan Stenn, and Mayank Varia.
8. IANA Considerations

This memo includes no request to IANA.

9. References

9.1. Normative References


9.2. Informative References


Authors’ Addresses

Aanchal Malhotra
Boston University
111 Cummington St
Boston, MA 02215
US

Email: aanchal4@bu.edu

Sharon Goldberg
Boston University
111 Cummington St
Boston, MA 02215
US

Email: goldbe@cs.bu.edu
Control Messages Protocol for Use with Network Time Protocol Version 4
draft-ietf-ntp-mode-6 cmds-05

Abstract

This document describes the structure of the control messages used with the Network Time Protocol. These control messages can be 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 a historic description of the control messages as described in RFC 1305 and any additional commands implemented in NTP.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 27, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect
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 historical record given their use within NTPv4.
1.1. Control Message Overview

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 larger than 576 octets; 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 and peer variables 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 a 16-bit association identifier 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 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 (ntpd) 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. Bit positions marked as zero are reserved and should always be transmitted as zero.
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:
<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>reserved</td>
</tr>
<tr>
<td>1</td>
<td>read status command/response</td>
</tr>
<tr>
<td>2</td>
<td>read variables command/response</td>
</tr>
<tr>
<td>3</td>
<td>write variables command/response</td>
</tr>
<tr>
<td>4</td>
<td>read clock variables command/response</td>
</tr>
<tr>
<td>5</td>
<td>write clock variables command/response</td>
</tr>
<tr>
<td>6</td>
<td>set trap address/port command/response</td>
</tr>
<tr>
<td>7</td>
<td>trap response</td>
</tr>
<tr>
<td>8</td>
<td>runtime configuration command/response</td>
</tr>
<tr>
<td>9</td>
<td>export configuration to file command/response</td>
</tr>
<tr>
<td>10</td>
<td>retrieve remote address stats command/response</td>
</tr>
<tr>
<td>11</td>
<td>retrieve ordered list command/response</td>
</tr>
<tr>
<td>12</td>
<td>request client-specific nonce command/response</td>
</tr>
<tr>
<td>13-30</td>
<td>reserved</td>
</tr>
<tr>
<td>31</td>
<td>unset trap address/port command/response</td>
</tr>
</tbody>
</table>

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 RFC 1305.

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.
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:

<table>
<thead>
<tr>
<th>LI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>no warning</td>
</tr>
<tr>
<td>01</td>
<td>insert second after 23:59:59 of the current day</td>
</tr>
<tr>
<td>10</td>
<td>delete second 23:59:59 of the current day</td>
</tr>
<tr>
<td>11</td>
<td>unsynchronized</td>
</tr>
</tbody>
</table>
Clock Source (Clock Src): This is a six-bit integer indicating the current synchronization source, with values coded as follows:

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

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

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:

<table>
<thead>
<tr>
<th>Peer Status</th>
<th>Meaning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>configured (peer.config)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>authentication enabled (peer.authenable)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>authentication okay (peer.authentic)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>reachability okay (peer.reach != 0)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>broadcast association</td>
<td></td>
</tr>
</tbody>
</table>

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 ntpd -q)
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:

<table>
<thead>
<tr>
<th>Clock Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>clock operating within nominals</td>
</tr>
<tr>
<td>1</td>
<td>reply timeout</td>
</tr>
<tr>
<td>2</td>
<td>bad reply format</td>
</tr>
<tr>
<td>3</td>
<td>hardware or software fault</td>
</tr>
<tr>
<td>4</td>
<td>propagation failure</td>
</tr>
<tr>
<td>5</td>
<td>bad date format or value</td>
</tr>
<tr>
<td>6</td>
<td>bad time format or value</td>
</tr>
<tr>
<td>7-15</td>
<td>reserved</td>
</tr>
</tbody>
</table>

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:
<table>
<thead>
<tr>
<th>Error Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>unspecified</td>
</tr>
<tr>
<td>1</td>
<td>authentication failure</td>
</tr>
<tr>
<td>2</td>
<td>invalid message length or format</td>
</tr>
<tr>
<td>3</td>
<td>invalid opcode</td>
</tr>
<tr>
<td>4</td>
<td>unknown association identifier</td>
</tr>
<tr>
<td>5</td>
<td>unknown variable name</td>
</tr>
<tr>
<td>6</td>
<td>invalid variable value</td>
</tr>
<tr>
<td>7</td>
<td>administratively prohibited</td>
</tr>
<tr>
<td>8-255</td>
<td>reserved</td>
</tr>
</tbody>
</table>

4. Commands

Commands consist of the header and optional data field shown in Figure 2. When present, the data field contains a list of identifiers or assignments in the form $<<\text{identifier}}>>[=<<\text{value}}>>],[<<\text{identifier}}>>[=<<\text{value}}>>],...$ where $<<\text{identifier}}>>$ is the ASCII name of a system or peer variable specified in RFC 5905 and $<<\text{value}}>>$ is expressed as a decimal, hexadecimal or string constant in the syntax of the C programming language. Where no ambiguity exists, the $<169>\text{sys.}<170>$ or $<169>\text{peer.<170>$ 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 four octets in the form $[n.n.n.n]$ where $n$ is in decimal notation and the brackets are optional. 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, while 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 $<169>.<170>$ 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 and values. 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. The reference implementation daemon requires authentication for this command.

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

Read MRU (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 an ordered list. 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. The reference implementation honors nonces which were issued less than 16 seconds prior. Regurgitation 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 a whitelist 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-2013-5211)). 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 [RFC2827].

- 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. CVE-2015-8139 describes the specific attack.

  * 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. CVE-2016-1548 describes the attack.

- 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. CVE-2016-1548 is one such example.

NTP best practices recommend configuring ntpd 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.
Recent work (reference needed) shows that 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.

7. 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.

8. Normative References


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.

Authors’ Addresses

Dr. David L. Mills
University of Delaware

Email: mills@udel.edu
Guidelines for Defining Packet Timestamps
draft-ietf-ntp-packet-timestamps-02

Abstract

This document specifies guidelines for defining binary packet timestamp formats in networking protocols at various layers. It also presents three recommended timestamp formats. The target audience of this memo includes network protocol designers. It is expected that a new network protocol that requires a packet timestamp will, in most cases, use one of the recommended timestamp formats. If none of the recommended formats fits the protocol requirements, the new protocol specification should specify the format of the packet timestamp according to the guidelines in this document.

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 December 26, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents
1. Introduction

Timestamps are widely used in network protocols for various purposes, including delay measurement, clock synchronization, and logging or reporting the time of an event.

Timestamps are represented in the RFC series in one of two forms: text-based timestamps, and packet timestamps. Text-based timestamps [RFC3339] are represented as user-friendly strings, and are widely used in the RFC series, for example in information objects and data models, e.g., [RFC5646], [RFC6991], and [RFC7493]. Packet timestamps, on the other hand, are represented by a compact binary
This memo presents guidelines for defining a packet timestamp format in network protocols. Three recommended timestamp formats are presented. It is expected that a new network protocol that requires a packet timestamp will, in most cases, use one of the recommended timestamp formats. If none of the recommended formats fits the protocol requirements, the new protocol specification should specify the format of the packet timestamp according to the guidelines in this document.

2. Terminology

2.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2.2. Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTP</td>
<td>Network Time Protocol [RFC5905]</td>
</tr>
<tr>
<td>PTP</td>
<td>Precision Time Protocol [IEEE1588]</td>
</tr>
<tr>
<td>TAI</td>
<td>International Atomic Time</td>
</tr>
<tr>
<td>UTC</td>
<td>Coordinated Universal Time</td>
</tr>
</tbody>
</table>

2.3. Terms used in this Document

- **Timestamp error**: The difference between the timestamp value at the device under test and the value of a reference clock at the same time instant.
- **Timestamp format**: The specification of a timestamp, which is represented by a set of attributes that unambiguously define the syntax and semantics of a timestamp.
- **Timestamp accuracy**: The mean over an ensemble of measurements of the timestamp error.
- **Timestamp precision**: The variation over an ensemble of measurements of the timestamp error.
Timestamp resolution: The minimal time unit used for representing the timestamp.

3. Packet Timestamp Specification Template

This memo recommends to use the timestamp formats defined in Section 4. In cases where these timestamp formats do not satisfy the protocol requirements, the timestamp specification should clearly state the reasons for defining a new format. Moreover, it is recommended to derive the new timestamp format from an existing timestamp format, either a timestamp format from this memo, or any other previously defined timestamp format.

The timestamp specification must unambiguously define the syntax and the semantics of the timestamp. The current section defines the minimum set of attributes, but it should be noted that in some cases additional attributes or aspects will need to be defined in the timestamp specification.

This section defines a template for specifying packet timestamps. A timestamp format specification MUST include at least the following aspects:

Timestamp syntax:

The structure of the timestamp field consists of:

+ Size: The number of bits (or octets) used to represent the packet timestamp field. If the timestamp is comprised of more than one field, the size of each field is specified.

Timestamp semantics:

+ Units: The units used to represent the timestamp. If the timestamp is comprised of more than one field, the units of each field are specified.

+ Resolution: The timestamp resolution; the resolution is equal to the timestamp field unit. If the timestamp consists of two or more fields using different time units, then the resolution is the smallest time unit.

+ Wraparound: The wraparound period of the timestamp; any further wraparound-related considerations should be described here.

+ Epoch: The origin of the timescale used for the timestamp; the moment in time used as a reference for the timestamp value. For example, the epoch may be based on a standard time scale, such as
UTC. Another example is a relative timestamp, in which the epoch is the time at which the device using the timestamp was powered up, and is not affected by leap seconds (see the next attribute).

Leap seconds: This subsection specifies whether the timestamp is affected by leap seconds. If the timestamp is affected by leap seconds, then it represents the time elapsed since the epoch minus the number of leap seconds that have occurred since the epoch.

4. Recommended Timestamp Formats

This memo defines a set of recommended timestamp formats. Defining a relatively small set of recommended formats enables significant reuse; for example, a network protocol may reuse the NTP or PTP timestamp format, allowing a straightforward integration with an NTP or a PTP-based timer. Moreover, since accurate timestamping mechanisms are often implemented in hardware, a new network protocol that reuses an existing timestamp format can be quickly deployed using existing hardware timestamping capabilities. This memo recommends to use one of the timestamp formats specified below.

Clearly, different network protocols may have different requirements and constraints, and consequently may use different timestamp formats. The choice of the specific timestamp format for a given protocol may depend on a various factors. A few examples of factors that may affect the choice of the timestamp format:

- Timestamp size: while some network protocols use a large timestamp field, in some cases there may be constraints with respect to the timestamp size, affecting the choice of the timestamp format.

- Resolution: the time resolution is another factor that may directly affect the selected timestamp format. A potentially important factor in this context is extensibility; it may be desirable to allow a timestamp format to be extensible to a higher resolution by extending the field. For example, the resolution of the NTP 32-bit timestamp format can be improved by extending it to the NTP 64-bit timestamp format in a straightforward way.

- Wraparound period: the length of the time interval in which the timestamp is unique may also be an important factor in choosing the timestamp format. Along with the timestamp resolution, these two factors determine the required number of bits in the timestamp.

- Common format for multiple protocols: if there are two or more network protocols that use timestamps and are often used together in typical systems, using a common timestamp format should be
preferred if possible. Specifically, if the network protocol that is being defined typically runs on a PC, then an NTP-based timestamp format may allow easier integration with an NTP-synchronized timer. In contrast, a protocol that is typically deployed on a hardware-based platform, may make better use of a PTP-based timestamp, allowing more efficient integration with a PTP-synchronized timer.

4.1. Using a Recommended Timestamp Format

A specification that uses one of the recommended timestamp formats should specify explicitly that this is a recommended timestamp format, and point to the relevant section in the current memo.

4.2. NTP Timestamp Formats

4.2.1. NTP 64-bit Timestamp Format

The Network Time Protocol (NTP) 64-bit timestamp format is defined in [RFC5905]. This timestamp format is used in several network protocols, including [RFC6374], [RFC4656], and [RFC5357]. Since this timestamp format is used in NTP, this timestamp format should be preferred in network protocols that are typically deployed in concert with NTP.

The format is presented in this section according to the template defined in Section 3.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Seconds                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Fraction                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Figure 1: NTP [RFC5905] 64-bit Timestamp Format
```

Timestamp field format:

- **Seconds**: specifies the integer portion of the number of seconds since the epoch.
  - + Size: 32 bits.
  - + Units: seconds.
Fraction: specifies the fractional portion of the number of seconds since the epoch.

+ Size: 32 bits.

+ Units: the unit is 2^(-32) seconds, which is roughly equal to 233 picoseconds.

Epoch:

The epoch is 1 January 1900 at 00:00 UTC.

Leap seconds:

This timestamp format is affected by leap seconds. The timestamp represents the number of seconds elapsed since the epoch minus the number of leap seconds.

Resolution:

The resolution is 2^(-32) seconds.

Wraparound:

This time format wraps around every 2^32 seconds, which is roughly 136 years. The next wraparound will occur in the year 2036.

4.2.2. NTP 32-bit Timestamp Format

The Network Time Protocol (NTP) 32-bit timestamp format is defined in [RFC5905]. This timestamp format is used in [I-D.ietf-ippm-initial-registry]. This timestamp format should be preferred in network protocols that are typically deployed in concert with NTP. The 32-bit format can be used either when space constraints do not allow the use of the 64-bit format, or when the 32-bit format satisfies the resolution and wraparound requirements.

The format is presented in this section according to the template defined in Section 3.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Seconds              |           Fraction            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: NTP [RFC5905] 32-bit Timestamp Format
Timestamp field format:

Seconds: specifies the integer portion of the number of seconds since the epoch.
+ Size: 16 bits.
+ Units: seconds.

Fraction: specifies the fractional portion of the number of seconds since the epoch.
+ Size: 16 bits.
+ Units: the unit is $2^{-16}$ seconds, which is roughly equal to 15.3 microseconds.

Epoch:

The epoch is 1 January 1900 at 00:00 UTC.

Leap seconds:

This timestamp format is affected by leap seconds. The timestamp represents the number of seconds elapsed since the epoch minus the number of leap seconds.

Resolution:

The resolution is $2^{-16}$ seconds.

Wraparound:

This time format wraps around every $2^{16}$ seconds, which is roughly 18 hours.

4.3. The PTP Truncated Timestamp Format

The Precision Time Protocol (PTP) [IEEE1588] uses an 80-bit timestamp format. The truncated timestamp format is a 64-bit field, which is the 64 least significant bits of the 80-bit PTP timestamp. Since this timestamp format is similar to the one used in PTP, this timestamp format should be preferred in network protocols that are typically deployed in PTP-capable devices.

The PTP truncated timestamp format was defined in [IEEE1588v1] and is used in several protocols, such as [RFC6374], [RFC7456], [RFC8186] and [ITU-T-Y.1731].
Figure 3: PTP [IEEE1588] Truncated Timestamp Format

Timestamp field format:

Seconds: specifies the integer portion of the number of seconds since the epoch.

+ Size: 32 bits.
+ Units: seconds.

Nanoseconds: specifies the fractional portion of the number of seconds since the epoch.

+ Size: 32 bits.
+ Units: nanoseconds. The value of this field is in the range 0 to \((10^9)-1\).

Epoch:

The PTP [IEEE1588] epoch is 1 January 1970 00:00:00 TAI, which is 31 December 1969 23:59:51.999918 UTC.

Leap seconds:

This timestamp format is not affected by leap seconds.

Resolution:

The resolution is 1 nanosecond.

Wraparound:

This time format wraps around every \(2^{32}\) seconds, which is roughly 136 years. The next wraparound will occur in the year 2106.
5. Synchronization Aspects

A specification that defines a new timestamp format or uses one of
the recommended timestamp formats should include a section on
Synchronization Aspects. Examples of such a section can be found in
Section 6).

The Synchronization Aspects section should specify all the
assumptions and requirements related to synchronization. For
example, the synchronization aspects may specify whether nodes
populating the timestamps should be synchronized among themselves,
and whether the timestamp is measured with respect to a central
reference clock such as an NTP server. If time is assumed to be
synchronized to a time standard such as UTC or TAI, it should be
specified in this section. Further considerations may be discussed
in this section, such as the required timestamp accuracy.

Another aspect that should be discussed in this section is leap
second [RFC5905] considerations. The timestamp specification
template (Section 3) specifies whether the timestamp is affected by
leap seconds. It is often the case that further details about leap
seconds will need to be defined in the Synchronization Aspects
section. Generally speaking, in a timekeeping system that considers
leap seconds, the system clock may be affected by a leap second in
one of three possible ways:

- The clock is turned backwards one second at the end of the leap
  second.
- The clock is frozen during the duration of the leap second.
- The clock is slowed down during and slightly after the duration of
  the leap second, until the new time value catches up.

The way leap seconds are handled depends on the synchronization
protocol, and is thus not specified in this document. However, if a
timestamp format is defined with respect to a timescale that is
affected by leap seconds, the Synchronization Aspects section should
specify how the use of leap seconds affects the timestamp usage.

6. Timestamp Use Cases

Packet timestamps are used in various network protocols. Typical
applications of packet timestamps include delay measurement, clock
synchronization, and others. The following table presents a (non-
exhaustive) list of protocols that use packet timestamps, and the
timestamp formats used in each of these protocols.
### Figure 4: Protocols that use Packet Timestamps

The rest of this section presents two hypothetic examples of network protocol specifications that use one of the recommended timestamp formats. The examples include the text that specifies the information related to the timestamp format.

#### 6.1. Example 1

**Timestamp:**

The timestamp format used in this specification is the NTP [RFC5905] 64-bit format, as specified in Section 4.2.1 of [I-D.ietf-ippm-initial-registry].

**Synchronization aspects:**

It is assumed that nodes that run this protocol are synchronized to UTC using a synchronization mechanism that is outside the scope of this document. In typical deployments this protocol will run on a machine that uses NTP [RFC5905] for synchronization. Thus, the timestamp may be derived from the NTP-synchronized clock, allowing the timestamp to be measured with respect to the clock of an NTP server. Since the NTP time format is affected by leap...
seconds, the current timestamp format is similarly affected. Thus, the value of a timestamp during or slightly after a leap second may be temporarily inaccurate.

6.2. Example 2

Timestamp:

The timestamp format used in this specification is the PTP [IEEE1588] Truncated format, as specified in Section 4.2.3 of [I-D.ietf-ntp-packet-timestamps].

Synchronization aspects:

It is assumed that nodes that run this protocol are synchronized among themselves. Nodes may be synchronized to a global reference time. Note that if PTP [IEEE1588] is used for synchronization, the timestamp may be derived from the PTP-synchronized clock, allowing the timestamp to be measured with respect to the clock of an PTP Grandmaster clock.

7. Packet Timestamp Control Field

In some cases it is desirable to have a control field that describes structure, format, content, and properties of timestamps. Control information about the timestamp format can be conveyed in some protocols using a dedicated control plane protocol, or may be made available at the management plane, for example using a YANG data model. An optional control field allows some of the control information to be attached to the timestamp.

An example of a packet timestamp control field is the Error Estimate field, defined by Section 4.1.2 in [RFC4656], which is used in OWAMP [RFC4656] and TWAMP [RFC5357].

This section defines high-level guidelines for defining packet timestamp control fields in network protocols that can benefit from such timestamp-related control information. The word 'requirements' is used in its informal context in this section.

7.1. High-level Control Field Requirements

A control field for packet timestamps must offer an adequate feature set and fulfill a series of requirements to be usable and accepted. The following list captures the main high-level requirements for timestamp fields.
1. Extensible Feature Set: protocols and applications depend on various timestamp characteristics. A timestamp control field must support a variable number of elements (components) that either describe or quantify timestamp-specific characteristics or parameters. Examples of potential elements include timestamp size, encoding, accuracy, leap seconds, reference clock identifiers, etc.

2. Size: Essential for an efficient use of timestamp control fields is the trade-off between supported features and control field size. Protocols and applications may select the specific control field elements that are needed for their operation from the set of available elements.

3. Composition: Applications may depend on specific control field elements being present in messages. The status of these elements may be either mandatory, conditional mandatory, or optional, depending on the specific application and context. A control field specification must support applications in conveying or negotiating (a) the set of control field elements along with (b) the status of any element (i.e., mandatory, conditional mandatory, or optional) by defining appropriate data structures and identity codes.

4. Category: Control field elements can characterize either static timestamp information (like, e.g., timestamp size in bytes and timestamp semantics: NTP 64 bit format) or runtime timestamp information (like, e.g., estimated timestamp accuracy at the time of sampling: 20 microseconds to UTC). For efficiency reason it may be meaningful to support separation of these two concepts: while the former (static) information is typically valid throughout a protocol session and may be conveyed only once, at session establishment time, the latter (runtime) information augments any timestamp instance and may cause substantial overhead for high-traffic protocols.

Proposals for timestamp control fields will be defined in separate documents and are out of scope of this memo.

8. IANA Considerations

This memo includes no request to IANA.

9. Security Considerations

A network protocol that uses a packet timestamp MUST specify the security considerations that result from using the timestamp. This
section provides an overview of some of the common security considerations of using timestamps.

Any metadata that is attached to control or data packets, and specifically packet timestamps, can facilitate network reconnaissance; by passively eavesdropping to timestamped packets an attacker can gather information about the network performance, and about the level of synchronization between nodes.

Timestamps can be spoofed or modified by on-path attackers, thus attacking the application that uses the timestamps. For example, if timestamps are used in a delay measurement protocol, an attacker can modify en route timestamps in a way that manipulates the measurement results. Integrity protection mechanisms, such as Hashed Message Authentication Codes (HMAC), can mitigate such attacks. The specification of an integrity protection mechanism is outside the scope of this document, as typically integrity protection will be defined on a per-network-protocol basis, and not specifically for the timestamp field.

Another potential threat that can have a similar impact is delay attacks. An attacker can maliciously delay some or all of the en route messages, with the same harmful implications as described in the previous paragraph. Mitigating delay attacks is a significant challenge; in contrast to spoofing and modification attacks, the delay attack cannot be prevented by cryptographic integrity protection mechanisms. In some cases delay attacks can be mitigated by sending the timestamped information through multiple paths, allowing to detect and to be resilient to an attacker that has access to one of the paths.

In many cases timestamping relies on an underlying synchronization mechanism. Thus, any attack that compromises the synchronization mechanism can also compromise protocols that use timestamping. Attacks on time protocols are discussed in detail in [RFC7384].

10. Acknowledgments

The authors thank Yaakov Stein, Greg Mirsky, Warner Losh, Rodney Cummings and other members of the NTP working group for many helpful comments. The authors gratefully acknowledge Harlan Stenn and the people from the Network Time Foundation for sharing their thoughts and ideas.
11. References

11.1. 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,

11.2. Informative References

[I-D.ietf-ippm-initial-registry]
Morton, A., Bagnulo, M., Eardley, P., and K. D’Souza,
"Initial Performance Metric Registry Entries", draft-ietf-
ippm-initial-registry-06 (work in progress), March 2018.

[I-D.ietf-ntp-packet-timestamps]
Mizrahi, T., Fabini, J., and A. Morton, "Guidelines for
Defining Packet Timestamps", draft-ietf-ntp-packet-
timestamps-01 (work in progress), March 2018.

[IEEE1588]
IEEE, "IEEE 1588 Standard for a Precision Clock
Synchronization Protocol for Networked Measurement and

[IEEE1588v1]
IEEE, "IEEE 1588 Standard for a Precision Clock
Synchronization Protocol for Networked Measurement and

[ITU-T-Y.1731]
ITU-T, "OAM functions and mechanisms for Ethernet based
Networks", 2013.

for High Performance", RFC 1323, DOI 10.17487/RFC1323, May

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet:
Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002,

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V.
Jacobson, "RTP: A Transport Protocol for Real-Time
Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550,


Authors’ Addresses

Tal Mizrahi
Marvell
6 Hamada st.
Yokneam
Israel

Email: talmi@marvell.com

Joachim Fabini
TU Wien
Gusshausstrasse 25/E389
Vienna 1040
Austria

Phone: +43 1 58801 38813
Fax: +43 1 58801 38898
Email: Joachim.Fabini@tuwien.ac.at
URI: http://www.tc.tuwien.ac.at/about-us/staff/joachim-fabini/

Al Morton
AT&T Labs
200 Laurel Avenue South
Middletown, NJ 07748
USA

Phone: +1 732 420 1571
Fax: +1 732 368 1192
Email: acmorton@att.com
URI: http://home.comcast.net/~acmacm/
RFC 5905 [RFC5905], section 7.3, "Packet Header Variables", defines the value of the REFID, the system peer for the responding host. In the past, for IPv4 associations the IPv4 address is used, and for IPv6 associations the first four octets of the MD5 hash of the IPv6 are used. There are at least three shortcomings to this approach, and this proposal will address the three so noted. One is that knowledge of the system peer is "abusable" information and should not be generally available. The second is that the four octet hash of the IPv6 address looks very much like an IPv4 address, and this is confusing. The third is that a growing number of low-stratum servers want to offer leap-smeared time to their clients, and there is no obvious way to know if a server is offering accurate time or leap-smeared time.

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 December 8, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.
1. Introduction

1.1. The REFID

The interpretation of a REFID is based on the stratum, as documented in RFC 5905 [RFC5905], section 7.3, "Packet Header Variables". The core reason for the REFID in the NTP Protocol is to prevent a degree-one timing loop, where server B decides to follow A as its time source, and A then decides to follow B as its time source.

At Stratum 2+, which will be the case if two servers A and B are exchanging timing information, then if server B follows A as its time
source, A’s address will be B’s REFID. When A uses IPv4, the default REFID is A’s IPv4 address. When A uses IPv6, the default REFID is a four-octet digest of A’s IPv6 address. Now, if A queries B for its time, then A will learn that B is using A as its time source by observing A’s address in the REFID field of the response packet sent by B. Thus, A will not select B as a potential time source, since this would cause a timing loop.

1.2. NOT-YOU REFID

This REFID mechanism, however, also allows a third-party C to learn that A is the time source that is being used by B. When A is using IPv4, C can learn this by querying B for its time, and observing that the REFID in B’s response is the IPv4 address of A. Meanwhile, when A is using IPv6, then C can again query B for its time, and then can use an offline dictionary attack to attempt to determine the IPv6 address that corresponds to the digest value in the response sent by B. C could construct the necessary dictionary by compiling a list of publicly accessible IPv6 servers. Remote attackers can use this technique to attempt to identify the time sources used by a target, and then send spoofed packets to the target or its time source in an attempt to disrupt time service, as was done e.g., in [NDSS16] or [CVE-2015-8138].

The REFID thus unnecessarily leaks information about a target’s time server to remote attackers. The best way to mitigate this vulnerability is to decouple the IP address of the time source from the REFID. To do this, a system can use an otherwise-impossible value for its REFID, called the NOT-YOU REFID value, when it believes that a querying system is not its time source.

The NOT-YOU REFID proposal is backwards-compatible and provides the most basic diagnostic information to third parties. It can be implemented by one peer in an NTP association without any changes to the other peer. This holds as long as responding NOT-YOU system can accurately detect when it’s getting a request from its system peer.

The NOT-YOU REFID proposal does have a small risk. Consider system A that returns the NOT-YOU REFID and system B that has two network interfaces B1 and B2. Suppose that system A is using system B as his time source, via network interface B1. Now suppose that system B queries system A for time via network interface B2. In this case, system A returns the NOT-YOU REFID value to system B, since system A does not realize that network interface B1 and B2 belong to the same system. In this case, system B might choose system A as its time source, and a degree-one timing loop will occur. In this case, however, the two systems will spiral into worse stratum positions with increasing root distances, and eventually the loop will break.
If any other systems are available as time servers, one of them will become the new system peer. However, until this happens the two spiraling systems will have degraded time quality.

1.3. IPv6 REFID

In an environment where all time queries made to a server can be trusted, an operator might well choose to expose the real REFID. RFC 5905 [RFC5905], section 7.3, "Packet Header Variables", explains how a remote system peer is converted to a REFID. It says:

> If using the IPv4 address family, the identifier is the four-octet IPv4 address. If using the IPv6 family, it is the first four octets of the MD5 hash of the IPv6 address. ...

However, the MD5 hash of an IPv6 address often looks like a valid IPv4 address. When this happens, an operator cannot tell if the REFID refers to an IPv6 address or and IPv4. Specifically, the NTP Project has received a report where the generated IPv6 hash decoded to the IPv4 address of a different machine on the system peer’s network.

This proposal offers a way for a system to generate a REFID for a IPv6 system peer that does not conflict with an IPv4-based REFID.

This proposal is not backwards-compatible. It SHOULD be implemented by both peers in an NTP association. In the scenario where A and B are peering using IPv6, where A is the system peer and does not understand IPv6 REFID, and B is subordinate and is using IPv6 REFID, A will not be able to determine that B is using A as its system peer and a degree-one timing loop can form.

If both peers implement the IPv6 REFID this situation cannot happen.

[If at least one of the peers implements the proposed I-DO protocol this situation cannot happen.]

1.4. Leap-Smear REFID

RFC 5905 [RFC5905] and earlier versions of NTP are the overwhelming method of distributing time on networks. Leap Seconds will continue to exist for a good number of years’ time, and since the timescale mandated by POSIX effectively ignores any instances where there are not 86,400 seconds’ time in a day something must be done to reliably synchronize clocks during the application of leap second corrections. One way to deal with the insertion of a leap second is to apply the leap second using a "smear", where the time reported by leap-second
aware servers is gradually adjusted so there is no major disruption to time synchronization when processing a leap second.

While the proper handling of leap seconds can be expected from up-to-date software and time servers, there are large numbers of out-of-date software installations and systems that are just not able to properly handle a leap second correction.

This proposal offers a way for a system to generate a REFID that indicates that the time being supplied in the NTP packet already contains an amount of leap smear correction, and what that amount is.

This proposal is backwards-compatible in all but poorly-designed NTP networks. The entire point of providing NTP servers that offer leap-smeared time in response to CLIENT requests is to provide smooth time to clients that are unable to properly handle leap seconds. If an operator is skilled enough to provide leap-smeared time to a subset of clients that cannot properly handle leap seconds, they can be expected to know enough to avoid using leap-smeared time between time servers that are expected to be able to properly handle leap seconds. Leap smears are expected to be implemented on a limited number of time servers where there is a base of client systems that cannot handle a leap second correction. Furthermore, even in a poorly-designed NTP network the "window of risk" lasts only as long as it takes for the leap second to be smeared.

1.5. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. The NOT-YOU REFID

2.1. Proposal

When enabled, this proposal allows the one-degree loop detection to work and useful diagnostic information to be provided to trusted partners while keeping potentially abusable information from being disclosed to ostensibly uninterested parties. It does this by returning the normal REFID to queries that come from trusted addresses or from an address that the current system believes is its time source (aka its "system peer"), and otherwise returning a special IP address that is interpreted to mean "not you". The "not you" IP address is 127.127.127.127 when the query is made from an IPv4 address, or when the query is made from an IPv6 address whose four-octet hash does not equal 127.127.127.127. The "not you" IP
address is 127.127.127.128 when the query is made from an address whose four-octet hash equals 127.127.127.127.

This mechanism is correct and transparent when the system responding with a NOT-YOU can accurately detect when it’s getting a timing query from its system peer. A querying system that uses IPv4 continues to check that its IPv4 address does not appear in the REFID before deciding whether to take time from the current system. A querying system that uses IPv6 continues to check that the four-octet hash of its IPv6 address does not appear in the REFID before deciding whether to take time from the current system.

3. Augmenting the IPv6 REFID Hash

3.1. Background

In a trusted network, the S2+ REFID is generated based on the network system peer. RFC 5905 [RFC5905] says:

If using the IPv4 address family, the identifier is the four-octet IPv4 address. If using the IPv6 family, it is the first four octets of the MD5 hash of the IPv6 address.

This means that the IPv4 representation of the IPv6 hash would be: b1.b2.b3.b4. The proposal is that the system MAY also use 255.b2.b3.b4 as its REFID. This reduces the risk of ambiguity, since addresses beginning with 255 are "reserved", and thus will not collide with valid IPv4 on the network.

When using the REFID to check for a timing loop for an IPv6 association, if the code that checks the first four-octets of the hash fails to match then the code must check again, using 0xFF as the first octet of the hash.

3.2. Potential Problems

There is a 1 in 16,777,216 chance that the REFID hashes of two IPv6 addresses will be identical, producing a false-positive loop detection. With a sufficient number of servers, the risk of this problem becomes a non-issue. [The use of the NOT-YOU REFID and/or the proposed "REFID Suggestion" or "I-DO" extension fields are ways to mitigate this potential situation.]

Unrealistically, if only two instances of NTP are communicating via IPv6 and system A implements this new IPv6 REFID hash and system B does not, system B will not be able to detect this loop condition. In this case, the two machines will slowly increase their stratum until they become unsynchronized. This situation is considered to be
unrealistic because, for this to happen, each system would have to have only the other system available as a time source, for example, in a misconfigured "orphan mode" setup. There is no risk of this happening in an NTP network with 3 or more time sources, or in a properly-configured "time island" setup.

3.3. Questions

Should we reference the REFID Suggestion and I-DO proposals here?

Should we ask IANA to allocate a pseudo Extension Field Type of 0xFFFF (for example) so the proposed "I-Do" exchange can report whether or not the "IPv6 REFID Hash" is supported?

4. The REFID sent to clients during a Leap-Smear

4.1. Background

This proposal offers a way for a system to generate a REFID that indicates that the time being supplied in the NTP packet already contains an amount of leap smear correction, and what that amount is.

4.2. Leap Smear REFID

RFC 5905 [RFC5905] defines the data type of NTP time values in Section 6, "Data Types":

All NTP time values are represented in twos-complement format, with bits numbered in big-endian (as described in Appendix A of [RFC0791]) fashion from zero starting at the left, or high-order, position. ... The 32 bit signed integer seconds portion and the 32 bit unsigned fractional seconds portion, or 32:32 format is:

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
|               Seconds               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
|            Fraction            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

NTP Timestamp Format (32:32)

This format provides coverage for 136 years’ time to a precision of 232 picoseconds. If a leap-second addition is being completely smeared just before before the stroke of the next POSIX second then
the smear correction will be (0,1). [That’s mathematical domain range notation - how to cite it?] If this was the only way to apply a leap smear correction then we could simply use an unsigned value to represent the correction. But while the first popular leap smear implementation applied the correction over an appropriate number of hours’ time before the actual leap second, so the system time was again correct at the stroke of 00:00, that meant that the difference between system time and UTC spent half of the duration of the smear application at [.5,1) "off" of correct time. The second popular implementation of the leap smear applied the first half-second correction before the stroke of 00:00 for a correction range of (0,.5] and the last half-second correction starting at the stroke of 00:00 for a [−.5,0) correction range. This also means we need a signed value to represent the amount of correction.

The REFID of a system that is supplying smeared time to client requests while leap-smeared correction is active would be 254.b1.b2.b3, where the three octets (b1, b2, and b3) are a 2:22 formatted value, yielding 2 signed bits of integer time and 22 bits of unsigned fractional subseconds, with a precision to 238 nanoseconds, or about a quarter of a microsecond. Signed time is needed to implement the mathematical range described in the previous paragraph.

[How should we cite the 2:22 notation? This is the same general format that we use for NTP timestamps.]

[Sharon says: I suggest adding a concrete example of the scheme, so that the above paragraph is easier to understand.]

The client is not expected to do anything with this information. Indeed, the whole point of offering smeared time is that there is reason to believe the clients are unable to properly handle a leap second correction. In this case, clients cannot be expected to do anything with data embedded in the REFID, either. However, monitoring systems that use tools that show a host’s system peer, like the ‘ntpq’ and ‘sntp’ programs in the reference implementation, [HMS: how to cite this?] can use this information to make sure that clients are following a leap-smearing server and can see fairly accurately what the smear is for each client.

Note that if an NTP server decides to offer smeared time corrections to clients, it SHOULD only offer this time in response to CLIENT time requests. An NTP server that is offering smeared time SHOULD NOT send smeared time in any peer exchanges. Also, system that sync their time via CLIENT requests SHOULD NOT be distributing time (smeared or otherwise) to other systems.
[Sharon asks: Consider a client that doesn’t know he is getting smeared time (b/c he is outdated etc). How is a this client supposed to know that he should not be distributing smeared time? Note that its perfectly normal for a stratum 2 server that gets his time via CLIENT requests from a stratum 1 server to then offer time to stratum 3 systems.]

We also note that during the application of a leap smear, the REFID from a system offering smeared time cannot provide detection of a timing loop. This is not expected to be a problem because time server systems are not expected to make CLIENT connections with each other, so they should not be receiving smeared time. [Sharon asks: I don’t understand this point, see my question above.] Moreso, if a time server is configured to make CLIENT connections to a server that offers smeared time, with the mechanism described here it can detect when it is getting smeared time, and either ignore time from that source, or "undo" the leap smear correction and use the corrected time for that sample.

This proposal is not an attempt to justify servers offering leap smeared time. It is only an attempt to make it easy and visible to identify when a server is offering or client is receiving smeared time, and provide the client a means to know the amount of smear correction as of the latest successful poll.

4.3. Questions

Should we ask IANA to allocate a pseudo Extension Field Type of 0xFFFE (for example) so the proposed "I-Do" exchange can report whether or not this server will offer leap smeared time in response to CLIENT time requests, identifying the amount of correction using the above REFID?

5. Acknowledgements

For the "not-you" REFID, we acknowledge useful discussions with Aanchal Malhotra and Matthew Van Gundy.

For the IPv6 REFID, we acknowledge Dan Mahoney (and perhaps others) for suggesting the idea of using an "impossible" first-octet value to indicate an IPv6 refid hash.

For the Leap Smear REFID, we acknowledge useful discussions with Juergen Perlinger.
6. IANA Considerations

This memo makes no requests of IANA.

7. Security Considerations

Many systems running NTP are configured to return responses to timing queries by default. These responses contain a REFID field, which generally reveals the address of the system’s time source if that source is an IPv4 address. This behavior can be exploited by remote attackers who wish to first learn the address of a target’s time source, and then attack the target and/or its time source. As such, the NOT-YOU REFID proposal is designed to harden NTP against these attacks by limiting the amount of information leaked in the REFID field.

Systems running NTP should reveal the identity of their system in peer in their REFID only when they are on a trusted network. The IPv6 REFID proposal provides one way to do this, when the system peer uses addresses in the IPv6 family.

8. References

8.1. Normative References


8.2. Informative References


Authors’ Addresses

Harlan Stenn
Network Time Foundation
P.O. Box 918
Talent, OR 97540
US

Email: stenn@nwtime.org

Sharon Goldberg
Boston University
111 Cummington St
Boston, MA 02215
US

Email: goldbe@cs.bu.edu
Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on January 2, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ................................. 3
   1.1. Objectives ................................ 3
   1.2. Protocol overview ......................... 4
2. Requirements Language ......................... 5
3. TLS profile for Network Time Security .......... 5
4. The NTS Key Establishment protocol ............. 6
   4.1. NTS-KE Record Types ...................... 8
       4.1.1. End of Message ....................... 8
       4.1.2. NTS Next Protocol Negotiation ....... 9
       4.1.3. Error ................................ 9
       4.1.4. Warning ............................. 10
       4.1.5. AEAD Algorithm Negotiation .......... 10
       4.1.6. New Cookie for NTPv4 ................. 11
   4.2. Key Extraction (generally) ................. 11
5. NTS Extension Fields for NTPv4 ................. 11
   5.1. Key Extraction (for NTPv4) ................ 11
   5.2. Packet structure overview ................. 12
   5.3. The Unique Identifier extension field .... 12
   5.4. The NTS Cookie extension field ............ 13
   5.5. The NTS Cookie Placeholder extension field . 13
   5.6. The NTS Authenticator and Encrypted Extension Fields extension field ..................... 13
6. Protocol details .............................. 15
7. Suggested format for NTS cookies ............... 18
8. IANA Considerations ........................... 20
9. Implementation Status ........................ 24
   9.1. Implementation PoC 1 ..................... 24
       9.1.1. Coverage ............................ 24
       9.1.2. Licensing ........................... 24
       9.1.3. Contact Information ................. 25
       9.1.4. Last Update ........................ 25
   9.2. Implementation PoC 2 ..................... 25
       9.2.1. Coverage ............................ 25
       9.2.2. Licensing ........................... 25
       9.2.3. Contact Information ................. 25
       9.2.4. Last Update ........................ 25
   9.3. Interoperability ........................ 25
10. Security considerations ....................... 26
    10.1. Avoiding DDoS amplification .............. 26
    10.2. Initial verification of server certificates .... 26
    10.3. Usage of NTP pools ...................... 27
    10.4. Delay attacks .......................... 27
    10.5. Random number generation ............... 28
1. Introduction

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

1.1. Objectives

The objectives of NTS are as follows:

- **Identity:** Through the use of the X.509 PKI, implementations may cryptographically establish the identity of the parties they are communicating with.

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

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

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

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

- **Unlinkability:** For mobile clients, NTS will not leak any information additional to NTP which would permit a passive adversary to determine that two packets sent over different networks came from the same client.
o Non-amplification: Implementations (especially server implementations) may avoid acting as DDoS amplifiers by never responding to a request with a packet larger than the request packet.

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

1.2. Protocol overview

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

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

The "NTS Extension Fields for NTPv4" are a collection of NTP extension fields for cryptographically securing NTPv4 using previously-established key material. They are suitable for securing client-server mode because the server can implement them without retaining per-client state, but on the other hand are suitable *only* for client-server mode because only the client, and not the server, is protected from replay.

The "NTS Key Establishment" protocol (NTS-KE) is a mechanism for establishing key material for use with the NTS Extension Fields for NTPv4. It uses TLS to exchange keys and negotiate some additional protocol options, but then quickly closes the TLS channel and permits the server to discard all associated state.
The typical protocol flow is as follows. The client connects to the server on the NTS TCP port and the two parties perform a TLS handshake. Via the TLS channel, the parties negotiate some additional protocol parameters and the server sends the client a supply of cookies. The parties use TLS key export [RFC5705] to extract key material which will be used in the next phase of the protocol. This negotiation takes only a single round trip, after which the server closes the connection and discards all associated state. At this point the NTS-KE phase of the protocol is complete.

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

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. TLS profile for Network Time Security

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

Since securing time protocols is (as of 2017) a novel application of TLS, no backward-compatibility concerns exist to justify using obsolete, insecure, or otherwise broken TLS features or versions. We therefore put forward the following requirements and guidelines, roughly representing 2017’s best practices.

Implementations MUST NOT negotiate TLS versions earlier than 1.2.

Implementations willing to negotiate more than one possible version of TLS SHOULD NOT respond to handshake failures by retrying with a downgraded protocol version. If they do, they MUST implement [RFC7507].

TLS clients MUST NOT offer, and TLS servers MUST NOT select, RC4 cipher suites. [RFC7465]
TLS 1.2 clients SHOULD offer, and TLS servers SHOULD accept, the TLS Renegotiation Indication Extension [RFC5746]. Regardless, they MUST NOT initiate or permit insecure renegotiation.

TLS 1.2 clients SHOULD offer, and TLS 1.2 servers SHOULD accept, the TLS Session Hash and Extended Master Secret Extension [RFC7627].

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

4. The NTS Key Establishment protocol

The NTS key establishment protocol is conducted via TCP port [[TBD1]]. The two endpoints carry out a TLS handshake in conformance with Section 3, with the client offering (via an ALPN [RFC7301] extension), and the server accepting, an application-layer protocol of "ntske/1". Immediately following a successful handshake, the client SHALL send a single request (as Application Data encapsulated in the TLS-protected channel), then the server SHALL send a single response followed by a TLS "Close notify" alert and then discard the channel state.

The client’s request and the server’s response each SHALL consist of a sequence of records formatted according to Figure 1. The sequence SHALL be terminated by a "End of Message" record, which has a Record Type of zero and a zero-length body. Furthermore, requests and non-error responses each SHALL include exactly one NTS Next Protocol Negotiation record.

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

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|C|         Record Type         |          Body Length          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1

The requirement that all NTS-KE messages be terminated by an End of Message record makes them self-delimiting.

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

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

- **Record Type:** A 15-bit integer in network byte order. The semantics of record types 0-5 are specified in this memo; additional type numbers SHALL be tracked through the IANA Network Time Security Key Establishment Record Types registry.

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

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

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

Figure 2 provides a schematic overview of the key exchange. It displays the protocol steps to be performed by the NTS client and server and record types to be exchanged.
4.1. NTS-KE Record Types

The following NTS-KE Record Types are defined.

4.1.1. End of Message

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

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

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

4.1.3. Error

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

Clients MUST NOT include Error records in their request. If clients receive a server response which includes an Error record, they MUST discard any negotiated key material and MUST NOT proceed to the Next Protocol.

The following error codes are defined.

Error code 0 means "Unrecognized Critical Record". The server MUST respond with this error code if the request included a record which the server did not understand and which had its Critical Bit set. The client SHOULD NOT retry its request without modification.

Error code 1 means "Bad Request". The server MUST respond with this error if, upon the expiration of an implementation-defined timeout, it has not yet received a complete and syntactically well-formed request from the client. This error is likely to be the result of a dropped packet, so the client SHOULD start over with a new TLS handshake and retry its request.
4.1.4. Warning

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

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

This memo defines no warning codes.

4.1.5. AEAD Algorithm Negotiation

The AEAD Algorithm Negotiation record has a Record Type number of 4. Its body consists of a sequence of unsigned 16-bit integers in network byte order, denoting Numeric Identifiers from the IANA AEAD registry [RFC5116]. The Critical Bit MAY be set.

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

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

Server implementations of NTS extension fields for NTPv4 (Section 5) MUST support AEAD_AES_SIV_CMAC_256 [RFC5297] (Numeric Identifier 15). That is, if the client includes AEAD_AES_SIV_CMAC_256 in its AEAD Algorithm Negotiation record, and the server accepts Protocol ID 0 (NTPv4) in its NTS Next Protocol Negotiation record, then the server’s AEAD Algorithm Negotiation record MUST NOT be empty.
4.1.6. New Cookie for NTPv4

The New Cookie for NTPv4 record has a Record Type number of 5. The contents of its body SHALL be implementation-defined and clients MUST NOT attempt to interpret them. See Section 7 for a suggested construction.

Clients MUST NOT send records of this type. Servers MUST send at least one record of this type, and SHOULD send eight of them, if they accept Protocol ID 0 (NTPv4) as a Next Protocol. The Critical Bit SHOULD NOT be set.

4.2. Key Extraction (generally)

Following a successful run of the NTS-KE protocol, key material SHALL be extracted according to RFC 5705 [RFC5705]. Inputs to the exporter function are to be constructed in a manner specific to the negotiated Next Protocol. However, all protocols which utilize NTS-KE MUST conform to the following two rules:

- The disambiguating label string MUST be "EXPORTER-network-time-security/1".
- The per-association context value MUST be provided, and MUST begin with the two-octet Protocol ID which was negotiated as a Next Protocol.

5. NTS Extension Fields for NTPv4

5.1. Key Extraction (for NTPv4)

Following a successful run of the NTS-KE protocol wherein Protocol ID 0 (NTPv4) is selected as a Next Protocol, two AEAD keys SHALL be extracted: a client-to-server (C2S) key and a server-to-client (S2C) key. These keys SHALL be computed according to RFC 5705 [RFC5705], using the following inputs.

- The disambiguating label string SHALL be "EXPORTER-network-time-security/1".
- The per-association context value SHALL consist of the following five octets:
  - The first two octets SHALL be zero (the Protocol ID for NTPv4).
  - The next two octets SHALL be the Numeric Identifier of the negotiated AEAD Algorithm, in network byte order.
The final octet SHALL be 0x00 for the C2S key and 0x01 for the S2C key.

Implementations wishing to derive additional keys for private or experimental use MUST NOT do so by extending the above-specified syntax for per-association context values. Instead, they SHOULD use their own disambiguating label string. Note that RFC 5705 provides that disambiguating label strings beginning with "EXPERIMENTAL" MAY be used without IANA registration.

5.2. Packet structure overview

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

- The usual 48-octet NTP header, which is authenticated but not encrypted.
- Some extension fields which are authenticated but not encrypted.
- An extension field which contains AEAD output (i.e., an authentication tag and possible ciphertext). The corresponding plaintext, if non-empty, consists of some extension fields which benefit from both encryption and authentication.
- Possibly, some additional extension fields which are neither encrypted nor authenticated. These are discarded by the receiver.

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

5.3. The Unique Identifier extension field

The Unique Identifier extension field has a Field Type of [[TBD2]]. When the extension field is included in a client packet (mode 3), its body SHALL consist of a string of octets generated uniformly at random. The string MUST be at least 32 octets long. When the extension field is included in a server packet (mode 4), its body SHALL contain the same octet string as was provided in the client packet to which the server is responding. Its use in modes other than client-server is not defined.

The Unique Identifier extension field provides the client with a cryptographically strong means of detecting replayed packets. It MAY also be used standalone, without NTS, in which case it provides the
client with a means of detecting spoofed packets from off-path attackers. Historically, NTP’s origin timestamp field has played both these roles, but for cryptographic purposes this is suboptimal because it is only 64 bits long and, depending on implementation details, most of those bits may be predictable. In contrast, the Unique Identifier extension field enables a degree of unpredictability and collision-resistance more consistent with cryptographic best practice.

5.4. The NTS Cookie extension field

The NTS Cookie extension field has a Field Type of [[TBD3]]. Its purpose is to carry information which enables the server to recompute keys and other session state without having to store any per-client state. The contents of its body SHALL be implementation-defined and clients MUST NOT attempt to interpret them. See Section 7 for a suggested construction. The NTS Cookie extension field MUST NOT be included in NTP packets whose mode is other than 3 (client) or 4 (server).

5.5. The NTS Cookie Placeholder extension field

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

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

5.6. The NTS Authenticator and Encrypted Extension Fields extension field

The NTS Authenticator and Encrypted Extension Fields extension field is the central cryptographic element of an NTS-protected NTP packet. Its Field Type is [[TBD5]] and the format of its body SHALL be as follows:

Nonce length: Two octets in network byte order, giving the length of the Nonce field and interpreted as an unsigned integer.
Nonce: A nonce as required by the negotiated AEAD Algorithm.

Ciphertext: The output of the negotiated AEAD Algorithm. The structure of this field is determined by the negotiated algorithm, but it typically contains an authentication tag in addition to the actual ciphertext.

Padding: several octets of padding, with every octet set to the number of padding octets included, e.g., "01", "02 02", or "03 03 03". Constraints on the number of padding octets included are enumerated below.

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

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

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

P: The plaintext SHALL consist of all (if any) NTP extension fields to be encrypted. The format of any such fields SHALL be in accordance with RFC 7822 [RFC7822], and if multiple extension fields are present they SHALL be joined by concatenation.

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

The number of padding octets included SHALL conform to the following constraints:

The number MUST be at least 1, so that the final octet of the extension field always gives the padding length.

The number MUST NOT be greater than 255, since high numbers are unrepresentable in a single octet.

The number MUST result in an extension field length which is legal per [RFC7822]. That is, the number of padding octets must be chosen so that the total length of the extension field (including the Field Type and Length subfields) is a multiple of 4 greater than or equal to 16, and greater than or equal to 28 if the extension field is the last one in the packet.
For mode 3 (client) packets only, the number MUST be at least
\( \text{MAX} (\text{MIN} (N_{\text{MAX}}, 16) - N_{\text{len}}, 0) + 4 \), where ‘\( N_{\text{len}} \)’ represents the
actual length of the nonce and \( N_{\text{MAX}} \) is, per [RFC5116], the
maximum permitted nonce length for the AEAD algorithm in use.
This constraint ensures that servers can always use an adequately
long nonce without causing the size of their response packet to
exceed the size of the request packet. Servers SHOULD enforce
this constraint by dropping client packets that do not conform to
it. Clients MUST NOT enforce it since it is not binding on mode 4
(server) packets to begin with.

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

6. Protocol details

A client sending an NTS-protected request SHALL include the following
extension fields as displayed in Figure 3:

- Exactly one Unique Identifier extension field, which MUST be
  authenticated, MUST NOT be encrypted, and whose contents MUST NOT
duplicate those of any previous request.

- Exactly one NTS Cookie extension field, which MUST be
  authenticated and MUST NOT be encrypted. The cookie MUST be one
  which the server previously provided the client; it may have been
  provided during the NTS-KE handshake or in response to a previous
  NTS-protected NTP request. To protect client’s privacy, the same
  cookie SHOULD NOT be included in multiple requests. If the client
does not have any cookies that it has not already sent, it SHOULD
re-run the NTS-KE protocol before continuing.

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

The client MAY include one or more NTS Cookie Placeholder extension
field, which MUST be authenticated and MAY be encrypted. The number
of NTS Cookie Placeholder extension fields that the client includes
SHOULD be such that if the client includes \( N \) placeholders and the
server sends back \( N+1 \) cookies, the number of unused cookies stored by
the client will come to eight. When both the client and server
adhere to all cookie-management guidance provided in this memo, the
number of placeholder extension fields will equal the number of
dropped packets since the last successful volley.
The client MAY include additional (non-NTS-related) extension fields, which MAY appear prior to the NTS Authenticator and Encrypted Extension Fields extension fields (therefore authenticated but not encrypted), within it (therefore encrypted and authenticated), or after it (therefore neither encrypted nor authenticated). In general, however, the server MUST discard any unauthenticated extension fields and process the packet as though they were not present. Servers MAY implement exceptions to this requirement for particular extension fields if their specification explicitly provides for such.

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

- Exactly one Unique Identifier extension field, which MUST be authenticated, MUST NOT be encrypted, and whose contents SHALL echo those provided by the client.
- Exactly one NTS Authenticator and Encrypted Extension Fields extension field, generated using the AEAD algorithm and S2C key recovered from the cookie provided by the client.
- One or more NTS Cookie extension fields, which MUST be encrypted and authenticated. The number of NTS Cookie extension fields included SHOULD be equal to, and MUST NOT exceed, one plus the number of valid NTS Cookie Placeholder extension fields included in the request.

We emphasize the contrast that NTS Cookie extension fields MUST NOT be encrypted when sent from client to server, but MUST be encrypted from sent from server to client. The former is necessary in order for the server to be able to recover the C2S and S2C keys, while the latter is necessary to satisfy the unlinkability goals discussed in Section 11.1. We emphasize also that "encrypted" means encapsulated within the the NTS Authenticator and Encrypted Extensions extension field. While the body of a NTS Cookie extension field will generally consist of some sort of AEAD output (regardless of whether the recommendations of Section 7 are precisely followed), this is not sufficient to make the extension field "encrypted".

The server MAY include additional (non-NTS-related) extension fields, which MAY appear prior to the NTS Authenticator and Encrypted Extension Fields extension field (therefore authenticated but not
encrypted), within it (therefore encrypted and authenticated), or after it (therefore neither encrypted nor authenticated). In general, however, the client MUST discard any unauthenticated extension fields and process the packet as though they were not present. Clients MAY implement exceptions to this requirement for particular extension fields if their specification explicitly provides for such.

If the server is unable to validate the cookie or authenticate the request, it SHOULD respond with a Kiss-o’-Death packet (see RFC 5905, Section 7.4) [RFC5905]) with kiss code "NTSN" (meaning "NTS NAK"). Such a response MUST include exactly one Unique Identifier extension field whose contents SHALL echo those provided by the client. It MUST NOT include any NTS Cookie or NTS Authenticator and Encrypted Extension Fields extension fields.

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

Upon receiving an NTS NAK, the client MUST verify that the Unique Identifier matches that of an outstanding request. If this check fails, the packet MUST be discarded without further processing. If this check passes, the client SHOULD wait until the next poll for a valid NTS-protected response and if none is received, discard all cookies and AEAD keys associated with the server which sent the NAK and initiate a fresh NTS-KE handshake.
- verify time request message
- generate time response message
- included NTPv4 extension fields
  o Unique Identifier EF
  o NTS Authentication and Encrypted Extension Fields EF
  - NTS Cookie EF
  - <NTS Cookie EF>
- transmit time request packet

Figure 3: NTS Time Synchronization Message

7. Suggested format for NTS cookies

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

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

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

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

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

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

- The AEAD algorithm negotiated during NTS-KE
- The S2C key
- The C2S key

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

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

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

8. IANA Considerations

IANA is requested to allocate two entries, identical except for the Transport Protocol, in the Service Name and Transport Protocol Port Number Registry as follows:

<table>
<thead>
<tr>
<th>Service Name: nts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport Protocol: tcp, udp</td>
</tr>
<tr>
<td>Assignee: IESG <a href="mailto:iesg@ietf.org">iesg@ietf.org</a></td>
</tr>
<tr>
<td>Contact: IETF Chair <a href="mailto:chair@ietf.org">chair@ietf.org</a></td>
</tr>
<tr>
<td>Description: Network Time Security</td>
</tr>
<tr>
<td>Reference: [[[this memo]]]</td>
</tr>
<tr>
<td>Port Number: [[[TBD1]]], selected by IANA from the user port range</td>
</tr>
</tbody>
</table>

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

| Protocol: Network Time Security Key Establishment, version 1 |
| Identification Sequence: 0x6E 0x74 0x73 0x6B 0x65 0x2F 0x31 ("ntske/1") |
| Reference: [[[this memo]]] |

IANA is requested to allocate the following entry in the TLS Exporter Label Registry:

<table>
<thead>
<tr>
<th>Value</th>
<th>DTLS-OK</th>
<th>Reference</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPORTER-network-time-security/1</td>
<td>Y</td>
<td>[[[this memo]]]</td>
<td></td>
</tr>
</tbody>
</table>

Franke, et al. Expires January 2, 2019
IANA is requested to allocate the following entry in the registry of NTP Kiss-o’-Death codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTSN</td>
<td>NTS NAK</td>
</tr>
</tbody>
</table>

IANA is requested to allocate the following entries in the NTP Extensions Field Types registry:

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Meaning</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>[[TBD2]]</td>
<td>Unique Identifier</td>
<td>[[this memo]]</td>
</tr>
<tr>
<td>[[TBD3]]</td>
<td>NTS Cookie</td>
<td>[[this memo]]</td>
</tr>
<tr>
<td>[[TBD4]]</td>
<td>NTS Cookie Placeholder</td>
<td>[[this memo]]</td>
</tr>
<tr>
<td>[[TBD5]]</td>
<td>NTS Authenticator and Encrypted Extension Fields</td>
<td>[[this memo]]</td>
</tr>
</tbody>
</table>

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

- **Type Number** (REQUIRED): An integer in the range 0-32767 inclusive.
- **Description** (REQUIRED): A short text description of the purpose of the field.
- **Set Critical Bit** (REQUIRED): One of "MUST", "SHOULD", "MAY", "SHOULD NOT", or "MUST NOT".
- **Reference** (REQUIRED): A reference to a document specifying the semantics of the record.

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

- 0-1023: IETF Review
- 1024-16383: Specification Required
- 16384-32767: Specification Required
16384-32767: Private and Experimental Use

Applications for new entries SHALL specify the contents of the Description, Set Critical Bit and Reference fields and which of the above ranges the Type Number should be allocated from. Applicants MAY request a specific Type Number, and such requests MAY be granted at the registrar’s discretion.

The initial contents of this registry SHALL be as follows:

<table>
<thead>
<tr>
<th>Field Number</th>
<th>Description</th>
<th>Critical</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>End of message</td>
<td>MUST</td>
<td>[[[this memo]]]</td>
</tr>
<tr>
<td>1</td>
<td>NTS next protocol negotiation</td>
<td>MUST</td>
<td>[[[this memo]]]</td>
</tr>
<tr>
<td>2</td>
<td>Error</td>
<td>MUST</td>
<td>[[[this memo]]]</td>
</tr>
<tr>
<td>3</td>
<td>Warning</td>
<td>MUST</td>
<td>[[[this memo]]]</td>
</tr>
<tr>
<td>4</td>
<td>AEAD algorithm negotiation</td>
<td>MAY</td>
<td>[[[this memo]]]</td>
</tr>
<tr>
<td>5</td>
<td>New cookie for NTPv4</td>
<td>SHOULD</td>
<td>[[[this memo]]]</td>
</tr>
<tr>
<td>16384-32767</td>
<td>Reserved for Private &amp; Experimental Use</td>
<td>MAY</td>
<td>[[[this memo]]]</td>
</tr>
</tbody>
</table>

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

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

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

Reference (RECOMMENDED): A reference to a relevant specification document. If no relevant document exists, a point-of-contact for questions regarding the entry SHOULD be listed here in lieu.

Applications for new entries in this registry SHALL specify all desired fields, and SHALL be granted upon approval by a Designated Expert. Protocol IDs 32768-65535 SHALL be reserved for Private or Experimental Use, and SHALL NOT be registered.
The initial contents of this registry SHALL be as follows:

<table>
<thead>
<tr>
<th>Protocol ID</th>
<th>Human-Readable Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Network Time Protocol version 4 (NTPv4)</td>
<td>[[[this memo]]]</td>
</tr>
<tr>
<td>32768-65535</td>
<td>Reserved for Private or Experimental Use</td>
<td>Reserved by [[[this memo]]]</td>
</tr>
</tbody>
</table>

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

- **Number** (REQUIRED): An integer in the range 0-65535 inclusive.
- **Description** (REQUIRED): A short text description of the condition.

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

- 0-1023: IETF Review
- 1024-32767: Specification Required
- 32768-65535: Private and Experimental Use

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

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unrecognized Critical Extension</td>
<td>[[[this memo]]]</td>
</tr>
<tr>
<td>1</td>
<td>Bad Request</td>
<td>[[[this memo]]]</td>
</tr>
</tbody>
</table>

The Network Time Security Warning Codes Registry SHALL initially be empty.
9. Implementation Status

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

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

9.1. Implementation PoC 1

Organization: Ostfalia University of Applied Science
Implementor: Martin Langer
Maturity: Proof-of-Concept Prototype

This implementation was used to verify consistency and to ensure completeness of this specification. It also demonstrate interoperability with NTP’s client-server mode messages.

9.1.1. Coverage

This implementation covers the complete specification.

9.1.2. Licensing

The code is released under a Apache License 2.0 license.

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

Contact Martin Langer: mart.langer@ostfalia.de

9.1.4. Last Update

The implementation was updated 3rd May 2018.

9.2. Implementation PoC 2

Organization: tbd

Implementor: Daniel Fox Franke

Maturity: Proof-of-Concept Prototype

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

9.2.1. Coverage

This implementation provides the client and the server for the initial TLS handshake and NTS key exchange. It provides the the client part of the NTS protected NTP messages.

9.2.2. Licensing

Public domain.

The source code is available at: https://github.com/dfoxfranke/nts-hackathon

9.2.3. Contact Information

Contact Daniel Fox Franke: dfoxfranke@gmail.com

9.2.4. Last Update

The implementation was updated 16th March 2018.

9.3. Interoperability

The Interoperability tests distinguished between NTS key exchange and NTS time exchange messages. For the NTS key exchange, interoperability between the two implementations has been verified successfully. Interoperability of NTS time exchange messages has been verified successfully for the case that PoC 1 represents the server and PoC 2 the client.
These tests successfully demonstrate that there are at least two running implementations of this draft which are able to interoperate.

10. Security considerations

10.1. Avoiding DDoS amplification

Certain non-standard and/or deprecated features of the Network Time Protocol enable clients to send a request to a server which causes the server to send a response much larger than the request. Servers which enable these features can be abused in order to amplify traffic volume in distributed denial-of-service (DDoS) attacks by sending them a request with a spoofed source IP. In recent years, attacks of this nature have become an endemic nuisance.

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

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

10.2. Initial verification of server certificates

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

Check whether the system time is in fact unreliable. If the system clock has previously been synchronized since last boot, then on operating systems which implement a kernel-based phase-
locked-loop API, a call to ntp_gettime() should show a maximum error less than NTP_PHASE_MAX. In this case, the clock SHOULD be considered reliable and certificates can be strictly validated.

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

Once the clock has been synchronized, periodically write the current system time to persistent storage. Do not accept any certificate whose notAfter field is earlier than the last recorded time.

Do not process time packets from servers if the time computed from them falls outside the validity period of the server’s certificate.

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

10.3. Usage of NTP pools

Additional standardization work and infrastructure development is necessary before NTS can be used with public NTP server pools. First, a scheme will need to be specified for determining what constitutes an acceptable certificate for a pool server, such as establishing a value required to be contained in its Extended Key Usage attribute, and how to determine, given the DNS name of a pool, what Subject Alternative Name to expect in the certificates of its members. Implementing any such specification will necessitate infrastructure work: pool organizers will need to act as certificate authorities, regularly monitor the behavior of servers to which certificates have been issued, and promptly revoke the certificate of any server found to be serving incorrect time.

10.4. Delay attacks

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

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

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

10.5. Random number generation

At various points in NTS, the generation of cryptographically secure random numbers is required. Whenever this draft specifies the use of random numbers, then cryptographically secure random number generation MUST be used. See [RFC4086] for guidelines concerning this topic.

11. Privacy Considerations

11.1. Unlinkability

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

NTS’s unlinkability objective is merely to not leak any additional data that could be used to link a device’s network address. NTS does not rectify legacy linkability issues that are already present in NTP. Thus, a client that requires unlinkability must also minimize information transmitted in a client query (mode 3) packet as described in the draft [I-D.ietf-ntp-data-minimization].
The unlinkability objective only holds for time synchronization traffic, as opposed to key exchange traffic. This implies that it cannot be guaranteed for devices that function not only as time clients, but also as time servers (because the latter can be externally triggered to send authentication data).

It should also be noted that it could be possible to link devices that operate as time servers from their time synchronization traffic, using information exposed in (mode 4) server response packets (e.g. reference ID, reference time, stratum, poll). Also, devices that respond to NTP control queries could be linked using the information revealed by control queries.

11.2. Confidentiality

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

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

12. Acknowledgements

The authors would like to thank Richard Barnes, Steven Bellovin, Scott Fluhrer, Sharon Goldberg, Russ Housley, Martin Langer, Miroslav Lichvar, Aanchal Malhotra, Dave Mills, Danny Mayer, Karen O’Donoghue, Eric K. Rescorla, Stephen Roettger, Kurt Roeckx, Kyle Rose, Rich Salz, Brian Sniffen, Susan Sons, Douglas Stebila, Harlan Stenn, Martin Thomson, and Richard Welty for contributions to this document and comments on the design of NTS.

13. References

13.1. Normative References

Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008,

[RFC5297]  Harkins, D., "Synthetic Initialization Vector (SIV)
Authenticated Encryption Using the Advanced Encryption
Standard (AES)", RFC 5297, DOI 10.17487/RFC5297, October

[RFC5705]  Rescorla, E., "Keying Material Exporters for Transport
Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705,

[RFC5746]  Rescorla, E., Ray, M., Dispensa, S., and N. Oskov,
"Transport Layer Security (TLS) Renegotiation Indication
Extension", RFC 5746, DOI 10.17487/RFC5746, February 2010,

Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010,

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

[RFC7301]  Friedl, S., Popov, A., Langley, A., and E. Stephan,
Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301,

[RFC7465]  Popov, A., "Prohibiting RC4 Cipher Suites", RFC 7465,
DOI 10.17487/RFC7465, February 2015,

Suite Value (SCSV) for Preventing Protocol Downgrade
Attacks", RFC 7507, DOI 10.17487/RFC7507, April 2015,
13.2. Informative References

[I-D.ietf-ntp-data-minimization]


Appendix A. Terms and Abbreviations

AEAD  Authenticated Encryption with Associated Data [RFC5116]

DDoS  Distributed Denial of Service

NTP  Network Time Protocol [RFC5905]

NTS  Network Time Security

TLS  Transport Layer Security

Authors’ Addresses

Daniel Fox Franke
Email: dfoxfranke@gmail.com
URI: https://www.dfranke.us

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

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

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

Phone: +49-(0)531-592-4471
Email: kristof.teichel@ptb.de
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 December 24, 2018.
1. Introduction


The data model covers configuration of system parameters of NTP, such as access rules, authentication and VPN Routing and Forwarding (VRF) binding, and also associations of NTP in different modes and 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.
2. NTP data model

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

module: ietf-ntp
  +--rw ntp!
    +--rw port?         uint16 {ntp-port}?
    +--rw refclock-master!
      +--rw master-stratum? ntp-stratum
    +--rw authentication
      +--rw auth-enabled?          boolean
      +--rw trusted-keys* [key-id]
        +--rw key-id
          -> /ntp/authentication/authentication-keys/key-id
      +--rw authentication-keys* [key-id]
        +--rw key-id       uint32
        +--rw algorithm?   identityref
        +--rw password?    ianach:crypt-hash
    +--rw access-rules
      +--rw access-rule* [access-mode]
        +--rw access-mode    access-modes
          +--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?
Internet-Draft                YANG for NTP                     June 2018

---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?       uint16 {ntp-port}?        
      +++-rw version?    ntp-version

++-ro associations* [address local-mode isConfigured]
  +++-ro address              inet:host
  +++-ro local-mode           association-modes
  +++-ro isConfigured         boolean
  +++-ro stratum?             ntp-stratum
  +++-ro refid?               union
  +++-ro authentication?
    ++-rw address              inet:host
    +++-rw prefer?     boolean
    +++-rw peer-interface? if:interface-ref
    +++-rw minpoll?    ntp-minpoll
    +++-rw maxpoll?    ntp-maxpoll
    +++-rw port?       uint16 {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? uint16 {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? uint16 {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? uint16 {ntp-port}?
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

- ntp-enabled
  - ntpEntStatusCurrentMode

YANG data nodes in /ntp
NTPv4-MIB objects

YANG data nodes in /ntp/associations
NTPv4-MIB objects

- address
  - ntpAssocAddressType
  - ntpAssocAddress

YANG NTP State Data Nodes and Related NTPv4-MIB Objects

- clock-state
  - clock-stratum
  - clock-refid
  - clock-precision
  - clock-offset
  - root-dispersion

- address
  - ntpAssocAddressType
  - ntpAssocAddress
  - ntpAssocStratum
  - ntpAssocRefId
  - ntpAssocOffset
  - ntpAssocStatusDelay
  - ntpAssocStatusDispersion
  - ntpAssocStatOutPkts
  - ntpAssocStatInPkts
  - ntpAssocStatProtocolError

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

<table>
<thead>
<tr>
<th>YANG data nodes in /ntp/</th>
<th>YANG data nodes in /system/ntp</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntp-enabled</td>
<td>enabled</td>
</tr>
<tr>
<td>unicast-configuration</td>
<td>server</td>
</tr>
<tr>
<td>unicast-configuration/address</td>
<td>server/transport/udp/address</td>
</tr>
<tr>
<td>unicast-configuration/port</td>
<td>server/transport/udp/port</td>
</tr>
<tr>
<td>unicast-configuration/type</td>
<td>server/association-type</td>
</tr>
<tr>
<td>unicast-configuration/iburst</td>
<td>server/iburst</td>
</tr>
<tr>
<td>unicast-configuration/prefer</td>
<td>server/prefer</td>
</tr>
</tbody>
</table>

YANG NTP Configuration Data Nodes and counterparts in RFC 7317

5. NTP YANG Module

```xml
<CODE BEGINS> file "ietf-ntp@2018-06-22.yang"
module ietf-ntp {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-ntp";
  prefix "ntp";

  import ietf-yang-types {
    prefix "yang";
    reference "RFC 6991";
  }

  import ietf-inet-types {
    prefix "inet";
    reference "RFC 6991";
  }

  import ietf-interfaces {
    prefix "if";
    reference "RFC 8343";
  }
}
```

Wu, et al. Expires December 24, 2018
import iana-crypt-hash {
    prefix "ianach";
    reference "RFC 7317";
}

import ietf-key-chain {
    prefix "key-chain";
    reference "RFC 8177";
}

import ietf-access-control-list {
    prefix "acl";
    reference "RFC XXXX";
}

import ietf-routing-types {
    prefix "rt-types";
    reference "RFC 8294";
}

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:   Eric Wu
                <mailto:eric.wu@huawei.com>
    Editor:   Anil Kumar S N
                <mailto:anil.ietf@gmail.com>
    Editor:   Yi Zhao
                <mailto:yi.z.zhao@ericsson.com>
    Editor:   Dhruv Dhody
                <mailto:dhruv.ietf@gmail.com>
    Editor:   Ankit Kumar Sinha
                <mailto:ankit.ietf@gmail.com>"

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

    Copyright (c) 2018 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"
typedef ntp-stratum {
  type uint8 {
    range "1..16";
  }
  description
  "The level of each server in the hierarchy is defined by
  a stratum number. Primary servers are assigned stratum
  one; secondary servers at each lower level are assigned
  stratum numbers one greater than the preceding level";
}

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

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

typedef ntp-maxpoll {
  type uint8 {
    range "4..17";
  }
}
default "10";
description
 "The maximul poll exponent for this NTP association.";
}
typedef access-modes {
 type enumeration {
 enum peer {
   value "0";
   description
   "Sets the fully access authority. Both time request and control query can be performed on the local NTP service, and the local clock can be synchronized to the remote server.";
 }
 enum server {
   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 to 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
   "Sets 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-modes {
  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 learnt this asso dynamically. This device may synchronize with its NTP peer.";
    }
    enum broadcast {
      value "3";
      description
       "Use broadcast mode(mode 5). This mode defines that its eigther 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
"This defines 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
  "This defines NTP sync state.";
}
Internet-Draft                YANG for NTP                     June 2018

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.";
}

/* feature */
feature ntp-port {
    description
        "Indicates that the device supports the configuration of
        the port for NTP.

        This is a ‘feature’, since many implementations do not
        support any port other than the default port.";
}

/* 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 password {
    type ianach:crypt-hash;
    description "Clear or encrypted mode for password text.";
}
}
grouping authentication-type-param {
    description
    "Authentication type.";
    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
        "Indicates the total number of packets sent.";
    }
    leaf packet-sent-fail {
        type yang:counter32;
        description
        "Indicates the number of times packet sending failed.";
    }
    leaf packet-received {
        type yang:counter32;
        description
        "Indicates the total number of packets received.";
    }
}
leaf packet-dropped {
    type yang:counter32;
    description
        "Indicates the number of packets dropped."
}

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 maximal poll interval used in this association.";
    }
    leaf port {
        if-feature ntp-port;
        type uint16 {
            range "123 | 1025..max";
        }
        default "123";
        description
            "Specify the port used to send NTP packets.";
    }
    leaf version {
        type ntp-version;
        description
            "NTP version.";
    }
}

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 address
             which result in clock synchronization.";
    }
    leaf associations-local-mode {
type leafref {
    path "/ntp:ntp/ntp:associations/ntp:local-mode";
}
description
"Indicates the association local-mode which result in clock synchronization.";

leaf associations-isConfigured {
    type leafref {
        path "/ntp:ntp/ntp:associations/" + "ntp:isConfigured";
    }
description
"Indicates the association was configured or dynamic which result in clock synchronization.";
}

/* Configuration data nodes */
container ntp {
presence "NTP is enable";
description "Configuration parameters for NTP.";
leaf port {
    if-feature ntp-port;
type uint16 {
        range "123 | 1025..max";
    }
default "123";
description "Specify the port used to send NTP packets.";
}
container refclock-master {
presence "NTP master clock is enable";
description "Configures the 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 trusted-keys {
  key "key-id";
  description
    "List of keys trusted by NTP."
  leaf key-id {
    type leafref {
      path "/ntp:ntp/ntp:authentication/
        + "ntp:authentication-keys/ntp:key-id";
    }
    description
      "The key trusted by NTP."
  }
}
list authentication-keys {
  key "key-id";
  uses authentication-key;
  description
    "List of authentication key."
}
container access-rules {
  description
    "Configuration of access rules."
  list access-rule {
    key "access-mode";
    description
      "List of access rules."
    leaf access-mode {
      type access-modes;
      description
        "NTP access mode."
    }
    leaf acl {
      type leafref {
        path "/acl:acls/acl:acl/acl:name";
      }
      description
        "NTP ACL."
    }
  }
}
container clock-state {
    config "false";
    description
    "Operational state of the NTP.";
}

container system-status {
    description
    "System status of NTP.";
    leaf clock-state {
        type ntp-clock-status;
        mandatory true;
        description "Indicates the state of system clock.";
    }
    leaf clock-stratum {
        type ntp-stratum;
        mandatory true;
        description
        "Indicates the stratum of the reference clock.";
    }
    leaf clock-refid {
        type union {
            type inet:ipv4-address;
            type binary {
                length "4";
            }
            type string {
                length "4";
            }
        }
        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.";
    }
}

uses association-ref {
    description
    "Reference to Association mode";
}

leaf nominal-freq {
    type decimal64 {
        fraction-digits 4;
    }
    mandatory true;
description
  "Indicates the nominal frequency of the
  local clock, in Hz."
}
leaf actual-freq {
  type decimal64 {
    fraction-digits 4;
  }
  mandatory true;
  description
    "Indicates the actual frequency of the
    local clock, in Hz."
}
leaf clock-precision {
  type uint8;
  mandatory true;
  description
    "Precision of the clock of this system
    in Hz. (prec=2^(-n))";
}
leaf clock-offset {
  type decimal64 {
    fraction-digits 4;
  }
  description
    "Offset of clock to synchronized peer,
    in milliseconds.";
}
leaf root-delay {
  type decimal64 {
    fraction-digits 2;
  }
  description
    "Total delay along path to root clock,
    in milliseconds.";
}
leaf root-dispersion {
  type decimal64 {
    fraction-digits 2;
  }
  description
    "Indicates the dispersion between the local clock
    and the master reference clock, in milliseconds.";
}
leaf reference-time {
  type yang:date-and-time;
  description
    "Indicates reference timestamp.";
leaf sync-state {
  type ntp-sync-state;
  mandatory true;
  description
    "Indicates the synchronization status of
    the local clock.";
}

list unicast-configuration {
  key "address type";
  description
    "list of unicast-configuration.";
  leaf address {
    type inet:host;
    description
      "The address of this association.";
  }
  leaf type {
    type unicast-configuration-type;
    description
      "Type for this NTP configuration";
  }
  container authentication{
    description
      "Authentication type.";
    uses authentication-type-param;
  }
  leaf prefer {
    type boolean;
    default "false";
    description
      "Whether this association is preferred.";
  }
  leaf burst {
    type boolean;
    default "false";
    description
      "Sends a series of packets instead of a single packet
      within each synchronization interval to achieve faster
      synchronization.";
  }
  leaf iburst {
    type boolean;
    default "false";
    description
      "Sends a series of packets instead of a single packet

within the initial synchronization interval to achieve
closer initial synchronization.

leaf source {
  type if:interface-ref;
  description
    "The interface whose ip address this association used
    as source address.";
}

uses common-attributes {
  description
    "Common attribute like port, version, min and max
    poll.";
}

list associations {
  key "address local-mode isConfigured";
  config "false";
  description
    "list of NTP association.";
  leaf address {
    type inet:host;
    description
      "The address of this association.";
  }
  leaf local-mode {
    type association-modes;
    description
      "Local mode for this NTP association.";
  }
  leaf isConfigured {
    type boolean;
    description
      "Whether this association is configured or
dynamically learnt.";
  }
  leaf stratum {
    type ntp-stratum;
    description
      "Indicates the stratum of the reference clock.";
  }
  leaf refid {
    type union {
      type inet:ipv4-address;
      type binary {
        length "4";
      }
      type string {

leaf reference-clock{
  type string;
  description
   "Reference clock type or address for the peer.";
}

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
   "Whether this association is preferred.";
}

leaf peer-interface {
  type if:interface-ref;
  description
   "The interface which is used for communication.";
}

uses common-attributes {
  description
   "Common attribute like port, version, min and
    max poll.";
}

leaf reach {
  type uint8;
  description
   "Indicates the reachability of the configured
    server or peer.";
}

leaf unreach {
  type uint8;
  description
   "Indicates the unreachability of the configured
    server or peer.";
}

leaf poll {
  type uint8;
  description
   "Indicates the polling interval for current,
    in seconds.";
}
leaf now {
    type uint32;
    description
    "Indicates the time since the NTP packet was not received or last synchronized, in seconds.";
}
leaf offset {
    type decimal64 {
        fraction-digits 4;
    }
    description
    "Indicates the offset between the local clock and the superior reference clock.";
}
leaf delay {
    type decimal64 {
        fraction-digits 2;
    }
    description
    "Indicates the delay between the local clock and the superior reference clock.";
}
leaf dispersion {
    type decimal64 {
        fraction-digits 2;
    }
    description
    "Indicates the dispersion between the local clock and the superior reference clock.";
}
leaf originate-time {
    type yang:date-and-time;
    description
    "Indicates packet originate timestamp(T1).";
}
leaf receive-time {
    type yang:date-and-time;
    description
    "Indicates packet receive timestamp(T2).";
}
leaf transmit-time {
    type yang:date-and-time;
    description
    "Indicates packet transmit timestamp(T3).";
}
leaf input-time {
    type yang:date-and-time;
    description
"Indicates packet input timestamp (T4).";
}

container ntp-statistics {
  description
  "Per Peer packet send and receive statistic.";
  uses statistics {
    description
    "NTP send and receive packet statistic.";
  }
}

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) of a broadcast packet.";
    }
    container authentication{ 
    description
      "Authentication type.";
    uses authentication-type-param;
  }
  uses common-attributes { 
    description
      "Common attribute like port, version, min and max poll.";
  }
}
container broadcast-client {
    presence
        "NTP broadcast-client is configured";
    description
        "Configuration of broadcast-client.";
}

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) of a multicast packet.";
    }
    container authentication{
        description
            "Authentication type.";
        uses authentication-type-param;
    }
    uses common-attributes {
        description
            "Common attribute like port, version, min and max poll.";
    }
}

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";
}  
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 type.";
    uses authentication-type-param;
  }
  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 attribute like port, version, min and max poll.";
}
reference
    "RFC 5905";
}
}

container ntp-statistics {
    config "false";
    description
        "Total NTP packet statistic.";
    uses statistics {
        description
            "NTP send and receive packet statistic.";
    }
}

6. Usage Example

6.1. Unicast association

Below is the example on 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 use the an IPv6 address (say 2001:DB8::1) in the "address" leaf with no change in any other data tree.

Below is the example on how to get unicast configuration

<get>
  <filter type="subtree">
      <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>
    </unicast-configuration>
  </ntp>
</data>
<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>
<origin-time>10-10-2017 07:33:55.253 Z+05:30</origin-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>

6.2. Refclock master

Below is the example on 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>

Below is the example on how to get reference clock configuration

<get>
  <filter type="subtree">
      <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>

6.3. Authentication configuration

Below is the example on how to enable authentication and configure authentication key 10 with mode as md5 and password 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>
          <password>abcd</password>
        </authentication-keys>
      </authentication>
    </ntp>
  </config>
</edit-config>

Below is the example on how to get authentication related configuration

<get>
  <filter type="subtree">
      <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>
        <password>abcd</password>
      </authentication-keys>
    </authentication>
  </ntp>
</data>
6.4. Access configuration

Below is the example on how to configure access type peer associated with acl 2000

```xml
<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>
```

Below is the example on how to get access related configuration

```xml
<get>
  <filter type="subtree">
      <sys:access-rules>
        <access-rule>
          <access-mode>peer</access-mode>
          <acl>2000</acl>
        </access-rule>
      </sys:access-rules>
    </sys:ntp>
  </filter>
</get>
```

6.5. Multicast configuration

Below is the example on 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>

Below is the example on how to get multicast-server related configuration
Below is the example on 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>

Below is the example on how to get multicast-client related configuration:

<get>
  <filter type="subtree">
      <sys:interfaces>
        <sys:interface>
          <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>
6.6. Manycast configuration

Below is the example on how to configure manycast-client with address as "224.1.1.1", port as 1025 and authentication keyid as 10

```xml
<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>
```

Below is the example on how to get manycast-client related configuration
<get>
  <filter type="subtree">
      <sys:interfaces>
        <sys:interface>
          <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>

Below is the example on 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>

Below is the example on how to get manycast-server related configuration

<get>
  <filter type="subtree">
      <sys:interfaces>
        <sys:interface>
          <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>
6.7. Clock state

Below is the example on how to get clock current state:

```xml
<get>
  <filter type="subtree">
      <sys:clock-state/>
    </sys:ntp>
  </filter>
</get>
```

```xml
<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>
```

6.8. Get all association

Below is the example on how to get all association present:
<get>
  <filter type="subtree">
      <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>2017-10-10 07:33:55.253 Z+05:30</originate-time>
      <receive-time>2017-10-10 07:33:55.258 Z+05:30</receive-time>
      <transmit-time>2017-10-10 07:33:55.300 Z+05:30</transmit-time>
      <input-time>2017-10-10 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>
6.9. Global statistic

Below is the example on how to get clock current state

```xml
<get>
  <filter type="subtree">
      <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>
```

7. 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.


Registrant Contact: The NETMOD WG of the IETF.

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

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

Name: ietf-ntp


Prefix: ntp

Reference: RFC XXXX
8. 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 [RFC5246].

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

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:
/ntp/associations - The entries in the list includes all active NTP associations of all modes. Unauthorized access to this needs to be curtailed.

9. Acknowledgments

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

10. References

10.1. Normative References

[I-D.ietf-netmod-acl-model]


10.2. Informative References


Authors' Addresses

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

Email: eric.wu@huawei.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

Dhruv Dhody
Huawei
Divyashree Techno Park, Whitefield
Bangalore, Kanataka 560066
India

Email: dhruv.ietf@gmail.com

Abstract

This document defines a YANG data model for the configuration of IEEE 1588-2008 devices and clocks, and also retrieval of the configuration information, data set and running states of IEEE 1588-2008 clocks. The YANG module in this document conforms to the Network Management Datastore Architecture (NMDA).

Status of this Memo

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

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

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

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

This Internet-Draft will expire on January 24, 2019.
1. Introduction

As a synchronization protocol, IEEE 1588-2008 [IEEE1588] is widely supported in the carrier networks, industrial networks, automotive networks, and many other applications. It can provide high precision time synchronization as fine as nano-seconds. The protocol depends on a Precision Time Protocol (PTP) engine to decide its own state automatically, and a PTP transportation layer to carry the PTP timing and various quality messages. The
configuration parameters and state data sets of IEEE 1588-2008 are numerous.

According to the concepts described in [RFC3444], IEEE 1588-2008 itself provides an information model in its normative specifications for the data sets (in IEEE 1588-2008 clause 8). Some standardization organizations including the IETF have specified data models in MIBs (Management Information Bases) for IEEE 1588-2008 data sets (e.g. [RFC8173], [IEEE8021AS]). These MIBs are typically focused on retrieval of state data using the Simple Network Management Protocol (SNMP), furthermore, configuration of PTP data sets is not considered in [RFC8173].

Some service providers and applications require that the management of the IEEE 1588-2008 synchronization network be flexible and more Internet-based (typically overlaid on their transport networks). Software Defined Network (SDN) is another driving factor, which demands an improved configuration capability of synchronization networks.

YANG [RFC7950] is a data modeling language used to model configuration and state data manipulated by network management protocols like the Network Configuration Protocol (NETCONF) [RFC6241]. A small set of built-in data types are defined in [RFC7950], and a collection of common data types are further defined in [RFC6991]. Advantages of YANG include Internet based configuration capability, validation, rollback and so on. All of these characteristics make it attractive to become another candidate modeling language for IEEE 1588-2008.

This document defines a YANG data model for the configuration of IEEE 1588-2008 devices and clocks, and retrieval of the state data of IEEE 1588-2008 clocks. The data model is based on the PTP data sets as specified in [IEEE1588]. The technology specific IEEE 1588-2008 information, e.g., those specifically implemented by a bridge, a router or a telecom profile, is out of scope of this document.

The YANG module in this document conforms to the Network Management Datastore Architecture (NMDA) [RFC8342].

When used in practice, network products in support of synchronization typically conform to one or more IEEE 1588-2008 profiles. Each profile specifies how IEEE 1588-2008 is used in a given industry (e.g. telecom, automotive) and application. A profile can require features that are optional in IEEE 1588-2008, and it can specify new features that use IEEE 1588-2008 as a foundation.
It is expected that the IEEE 1588-2008 YANG module be used as follows:

- The IEEE 1588-2008 YANG module can be used as-is for products that conform to one of the default profiles specified in IEEE 1588-2008.

- When the IEEE 1588 standard is revised (e.g. the IEEE 1588 revision in progress at the time of writing this document), it will add some new optional features to its data sets. The YANG module of this document MAY be revised and extended to support these new features. Moreover, the YANG "revision" SHOULD be used to indicate changes to the YANG module under such a circumstance.

- A profile standard based on IEEE 1588-2008 may create a dedicated YANG module for its profile. The profile’s YANG module SHOULD use YANG "import" to import the IEEE 1588-2008 YANG module as its foundation. Then the profile’s YANG module SHOULD use YANG "augment" to add any profile-specific enhancements.

- A product that conforms to a profile standard can also create its own YANG module. The product’s YANG module SHOULD "import" the profile’s module, and then use YANG "augment" to add any product-specific enhancements.

1.1. Conventions used in this document

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.

1.2. Terminology

Most terminologies used in this document are extracted from [IEEE1588].

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC</td>
<td>Boundary Clock, see Section 3.1.3 of [IEEE1588]</td>
</tr>
<tr>
<td>DS</td>
<td>Data Set</td>
</tr>
<tr>
<td>E2E</td>
<td>End-to-End</td>
</tr>
<tr>
<td>EUI</td>
<td>Extended Unique Identifier</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
</tbody>
</table>
2. IEEE 1588-2008 YANG Model hierarchy

This section describes the hierarchy of an IEEE 1588-2008 YANG module. Query and configuration of device wide or port specific configuration information and clock data set are described for this version.

Query and configuration of clock information include:

(Note: The attribute names are consistent with IEEE 1588-2008, but changed to the YANG style, i.e., using all lower-case, with dashes between words.)

- Clock data set attributes in a clock node, including: current-ds, parent-ds, default-ds, time-properties-ds, and transparent-clock-default-ds.
- Port-specific data set attributes, including: port-ds and transparent-clock-port-ds.

The readers are assumed to be familiar with IEEE 1588-2008. As all PTP terminologies and PTP data set attributes are described in details in IEEE 1588-2008 [IEEE1588], this document only outlines each of them in the YANG module.

A simplified YANG tree diagram [RFC8340] representing the data model is typically used by YANG modules. This document uses the same tree diagram syntax as described in [RFC8340].

module: ietf-ptp
  +--rw ptp
    |  +--rw instance-list* [instance-number]
    |     |  +--rw instance-number      uint32
    |     +--rw default-ds
    |        |  +--rw two-step-flag?    boolean
    |        |  +--ro clock-identity?   clock-identity-type
    |        +--rw number-ports? uint16
    |        +--rw clock-quality
    |            |  +--rw offset-scaled-log-variance? uint16
    |            +--rw priority1?        uint8
    |            +--rw priority2?        uint8
    |            +--rw domain-number?    uint8
    |            +--rw slave-only?       boolean
    |    +--rw current-ds
    |        |  +--rw steps-removed?        uint16
    |        |  +--rw offset-from-master?   time-interval-type
    |        +--rw mean-path-delay?   time-interval-type
    |    +--rw parent-ds
    |        |  +--rw parent-port-identity
    |        |        |  +--rw clock-identity?   clock-identity-type
    |        |        +--rw port-number?      uint16
    |        |        +--rw parent-stats?      boolean
    |        |        +--rw observed-parent-offset-scaled-log-variance? uint16
    |        |        +--rw observed-parent-clock-phase-change-rate?    int32
    |        |        +--rw grandmaster-identity?            clock-identity-type
    |        |        +--rw grandmaster-clock-quality
    |        |        |  +--rw clock-class?                uint8
    |        |        |  +--rw clock-accuracy?              uint8
    |        |        |  +--rw offset-scaled-log-variance?   uint16
    |        |        +--rw grandmaster-priority1?       uint8
rw grandmaster-priority2?         uint8

rw time-properties-ds
  rw current-utc-offset-valid?   boolean
  rw current-utc-offset?         int16
  rw leap59?                     boolean
  rw leap61?                     boolean
  rw time-traceable?             boolean
  rw frequency-traceable?        boolean
  rw ptp-timescale?              boolean
  rw time-source?                uint8

rw port-ds-list* [port-number]
  rw port-number                  uint16
  rw port-state?                  port-state-enumeration
  rw underlying-interface?        if:interface-ref
  rw log-min-delay-req-interval?  int8
  rw peer-mean-path-delay?        time-interval-type
  rw log-announce-interval?       int8
  rw announce-receipt-timeout?    uint8
  rw log-sync-interval?           int8
  rw delay-mechanism?             delay-mechanism-enumeration
  rw log-min-pdelay-req-interval? int8
  rw version-number?              uint8

rw transparent-clock-default-ds
  ro clock-identity?             clock-identity-type
  rw number-ports?               uint16
  rw delay-mechanism?            delay-mechanism-enumeration
  rw primary-domain?             uint8

rw transparent-clock-port-ds-list* [port-number]
  rw port-number                  uint16
  rw log-min-pdelay-req-interval? int8
  rw faulty-flag?                 boolean
  rw peer-mean-path-delay?        time-interval-type
2.1. Interpretations from IEEE 1588 Working Group

The preceding model and the associated YANG module have some subtle differences from the data set specifications of IEEE Std 1588-2008. These differences are based on interpretation from the IEEE 1588 Working Group, and are intended to provide compatibility with future revisions of the IEEE 1588 standard.

In IEEE Std 1588-2008, a physical product can implement multiple PTP clocks (i.e., ordinary, boundary, or transparent clock). As specified in 1588-2008 subclause 7.1, each of the multiple clocks operates in an independent domain. However, the organization of multiple PTP domains was not clear in the data sets of IEEE Std 1588-2008. This document introduces the concept of PTP instance as described in the new revision of IEEE 1588. The instance concept is used exclusively to allow for optional support of multiple domains. The instance number has no usage within PTP messages.

Based on statements in IEEE 1588-2008 subclauses 8.3.1 and 10.1, most transparent clock products have interpreted the transparent clock data sets to reside as a singleton at the root level of the managed product, and this YANG model reflects that location.

2.2. Configuration and state

The information model of IEEE Std 1588-2008 classifies each member in PTP data sets as one of the following:

- Configurable: Writable by management.
- Dynamic: Read-only to management, and the value is changed by 1588 protocol operation.
- Static: Read-only to management, and the value typically does not change.

For details on the classification of each PTP data set member, refer to the IEEE Std 1588-2008 specification for that member.

Under certain circumstances, the classification of an IEEE 1588 data set member may change for a YANG implementation, for example, a configurable member needs to be changed to read-only. In such a case, an implementation MAY choose to return a warning upon writing to a read-only member, or use the deviation mechanism to develop a new deviation model as described in Section 7.20.3 of [RFC7950].
This module imports typedef "interface-ref" from [RFC8343]. Most attributes are based on the information model defined in [IEEE1588], but their names are adapted to the YANG style of naming.

<CODE BEGINS> file "ietf-ptp@2018-07-24.yang"
module ietf-ptp {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-ptp";
  prefix "ptp";
  import ietf-interfaces {
    prefix if;
    reference "RFC8343: A YANG Data Model for Interface Management";
  }
  organization "IETF TICTOC Working Group";
  contact
    WG Web: http://tools.ietf.org/wg/tictoc/
    WG List: <mailto:tictoc@ietf.org>
    Editor: Yuanlong Jiang
    <mailto:jiangyuanlong@huawei.com>
    Editor: Rodney Cummings
    <mailto:rodney.cummings@ni.com>
  description "This YANG module defines a data model for the configuration of IEEE 1588-2008 clocks, and also for retrieval of the state data of IEEE 1588-2008 clocks.";
  revision "2018-07-24" {
    description "Initial version";
    reference "RFC XXXX: YANG Data Model for IEEE 1588-2008";
    //Note to RFC Editor: update RFC XXXX to the actual RFC number
  }
  typedef delay-mechanism-enumeration {
    type enumeration {
      enum e2e {
        value 1;
        description "The port uses the delay request-response mechanism.";
      }
    }
  

enum p2p {
  value 2;
  description
    "The port uses the peer delay mechanism.";
}
enum disabled {
  value 254;
  description
    "The port does not implement any delay mechanism.";
}


description
  "The propagation delay measuring option used by the
  port. Values for this enumeration are specified
  by the IEEE 1588 standard exclusively.";
reference
  "IEEE Std 1588-2008: 8.2.5.4.4";
}

typedef port-state-enumeration {
  type enumeration {
    enum initializing {
      value 1;
      description
        "The port is initializing its data sets, hardware, and
        communication facilities.";
    }
    enum faulty {
      value 2;
      description
        "The port is in the fault state.";
    }
    enum disabled {
      value 3;
      description
        "The port is disabled, and is not communicating PTP
        messages (other than possibly PTP management
        messages).";
    }
    enum listening {
      value 4;
      description
        "The port is listening for an Announce message.";
    }
    enum pre-master {
      value 5;
      description
        "The port is ready to synchronize with another
        port.";
    }
  };
}
"The port is in the pre-master state."
}
enum master {
  value 6;
  description
  "The port is behaving as a master port."
}
enum passive {
  value 7;
  description
  "The port is in the passive state."
}
enum uncalibrated {
  value 8;
  description
  "A master port has been selected, but the port is still in the uncalibrated state."
}
enum slave {
  value 9;
  description
  "The port is synchronizing to the selected master port."
}

description
"The current state of the protocol engine associated with the port. Values for this enumeration are specified by the IEEE 1588 standard exclusively."
reference
"IEEE Std 1588-2008: 8.2.5.3.1, 9.2.5"
}
typedef time-interval-type {
  type int64;
  description
  "Derived data type for time interval, represented in units of nanoseconds and multiplied by 2^16"
  reference
  "IEEE Std 1588-2008: 5.3.2"
}
typedef clock-identity-type {
  type binary {
    length "8";
  }
  description
grouping clock-quality-grouping {
  description "Derived data type for quality of a clock, which contains
clockClass, clockAccuracy and offsetScaledLogVariance.";
  reference "IEEE Std 1588-2008: 5.3.7";
  leaf clock-class {
      type uint8;
      default 248;
      description "The clockClass denotes the traceability of the time
or frequency distributed by the clock.";
  }
  leaf clock-accuracy {
      type uint8;
      description "The clockAccuracy indicates the expected accuracy
of the clock.";
  }
  leaf offset-scaled-log-variance {
      type uint16;
      description "The offsetScaledLogVariance provides an estimate of
the variations of the clock from a linear timescale
when it is not synchronized to another clock
using the protocol.";
  }
}

container ptp {
  description "The PTP struct containing all attributes of PTP data set,
other optional PTP attributes can be augmented as well.";
  list instance-list {
      key "instance-number";
  }
}
description
"List of one or more PTP data sets in the device (see IEEE Std 1588-2008 subclause 6.3).
Each PTP data set represents a distinct instance of
PTP implementation in the device (i.e., distinct
Ordinary Clock or Boundary Clock)."

leaf instance-number {
  type uint32;
  description
  "The instance number of the current PTP instance.
  This instance number is used for management purposes
  only. This instance number does not represent the PTP
domain number, and is not used in PTP messages.";
}

container default-ds {
  description
  "The default data set of the clock (see IEEE Std
  1588-2008 subclause 8.2.1).";

  leaf two-step-flag {
    type boolean;
    description
    "When set to true, the clock is a two-step clock;
    otherwise, the clock is a one-step clock.";
  }

  leaf clock-identity {
    type clock-identity-type;
    config false;
    description
    "The clockIdentity of the local clock";
  }

  leaf number-ports {
    type uint16;
    description
    "The number of PTP ports on the instance.";
  }

  container clock-quality {
    description
    "The clockQuality of the local clock.";
    uses clock-quality-grouping;
  }
}
leaf priority1 {
    type uint8;
    description
        "The priority1 attribute of the local clock.";
}

leaf priority2{
    type uint8;
    description
        "The priority2 attribute of the local clock.";
}

leaf domain-number {
    type uint8;
    description
        "The domain number of the current syntonization domain.";
}

leaf slave-only {
    type boolean;
    description
        "When set to true, the clock is a slave-only clock.";
}

container current-ds {
    description
        "The current data set of the clock (see IEEE Std 1588-2008 subclause 8.2.2).";

    leaf steps-removed {
        type uint16;
        default 0;
        description
            "The number of communication paths traversed between the local clock and the grandmaster clock.";
    }

    leaf offset-from-master {
        type time-interval-type;
        description
            "The current value of the time difference between a master and a slave clock as computed by the slave.";
    }
}
leaf mean-path-delay {
  type time-interval-type;
  description
    "The current value of the mean propagation time between
     a master and a slave clock as computed by the slave.";
}

container parent-ds {
  description
    "The parent data set of the clock (see IEEE Std 1588-2008
     subclause 8.2.3).";

carrier parent-port-identity {
  description
    "The portIdentity of the port on the master, it
     contains two members: clockIdentity and portNumber.";
  reference
    "IEEE Std 1588-2008: 5.3.5";

carrier clock-identity {
  type clock-identity-type;
  description
    "Identity of the clock";
}

carrier port-number {
  type uint16;
  description
    "Port number";
}
}

leaf parent-stats {
  type boolean;
  default false;
  description
    "When set to true, the values of
     observedParentOffsetScaledLogVariance and
     observedParentClockPhaseChangeRate of parentDS
     have been measured and are valid.";
}

leaf observed-parent-offset-scaled-log-variance {
}
leaf observed-parent-clock-phase-change-rate {
  type int32;
  description
  "An estimate of the parent clock's phase change rate
   as observed by the slave clock.";
}

leaf grandmaster-identity {
  type clock-identity-type;
  description
  "The clockIdentity attribute of the grandmaster clock.";
}

canterior grandmaster-clock-quality {
  description
  "The clockQuality of the grandmaster clock.";
  uses clock-quality-grouping;
}

leaf grandmaster-priority1 {
  type uint8;
  description
  "The priority1 attribute of the grandmaster clock.";
}

leaf grandmaster-priority2 {
  type uint8;
  description
  "The priority2 attribute of the grandmaster clock.";
}


canterior time-properties-ds {
  description
  "The timeProperties data set of the clock (see
   IEEE Std 1588-2008 subclause 8.2.4).";

  leaf current-utc-offset-valid {
    type boolean;
  }
}
leaf current-utc-offset {
    when "../current-utc-offset-valid='true'"
    type int16;
    description
        "The offset between TAI and UTC when the epoch of the
         PTP system is the PTP epoch in units of seconds, i.e.,
         when ptp-timescale is TRUE; otherwise, the value has
         no meaning.";
}

leaf leap59 {
    type boolean;
    description
        "When set to true, the last minute of the current UTC
day contains 59 seconds.";
}

leaf leap61 {
    type boolean;
    description
        "When set to true, the last minute of the current UTC
day contains 61 seconds.";
}

leaf time-traceable {
    type boolean;
    description
        "When set to true, the timescale and the
currentUtcOffset are traceable to a primary
reference.";
}

leaf frequency-traceable {
    type boolean;
    description
        "When set to true, the frequency determining the
timescale is traceable to a primary reference.";
}

leaf ptp-timescale {
    type boolean;
    description
"When set to true, the clock timescale of the
grandmaster clock is PTP; otherwise, the timescale is
ARB (arbitrary).";
}

leaf time-source {
  type uint8;
  description
  "The source of time used by the grandmaster clock.";
}

list port-ds-list {
  key "port-number";
  description
  "List of port data sets of the clock (see IEEE Std
1588-2008 subclause 8.2.5).";

  leaf port-number {
    type uint16;
    description
    "Port number.
    The data sets (i.e., information model) of IEEE Std
1588-2008 specify a member portDS.portIdentity, which
uses a typed struct with members clockIdentity and
portNumber.

    In this YANG data model, portIdentity is not modeled
in the port-ds-list, however, its members are provided
as follows:
    portIdentity.portNumber is provided as this port-
number leaf in port-ds-list; and
    portIdentity.clockIdentity is provided as the clock-
identity leaf in default-ds of the instance
(i.e., ../../default-ds/clock-identity).";
  }

  leaf port-state {
    type port-state-enumeration;
    default "initializing";
    description
    "Current state associated with the port.";
  }

  leaf underlying-interface {

type if:interface-ref;
description
"Reference to the configured underlying interface that
is used by this PTP Port (see RFC 8343).";
}

leaf log-min-delay-req-interval {
  type int8;
  description
  "The base-two logarithm of the minDelayReqInterval
  (the minimum permitted mean time interval between
  successive Delay_Req messages).";
}

leaf peer-mean-path-delay {
  type time-interval-type;
  default 0;
  description
  "An estimate of the current one-way propagation delay
  on the link when the delayMechanism is P2P; otherwise,
  it is zero.";
}

leaf log-announce-interval {
  type int8;
  description
  "The base-two logarithm of the mean
  announceInterval (mean time interval between
  successive Announce messages).";
}

leaf announce-receipt-timeout {
  type uint8;
  description
  "The number of announceInterval that have to pass
  without receipt of an Announce message before the
  occurrence of the event ANNOUNCE_RECEIPT_TIMEOUT_ EXPIRES.";
}

leaf log-sync-interval {
  type int8;
  description
  "The base-two logarithm of the mean SyncInterval
  for multicast messages. The rates for unicast
  transmissions are negotiated separately on a per port
  basis and are not constrained by this attribute.";
leaf delay-mechanism {
  type delay-mechanism-enumeration;
  description
  "The propagation delay measuring option used by the
  port in computing meanPathDelay.";
}

leaf log-min-pdelay-req-interval {
  type int8;
  description
  "The base-two logarithm of the
  minPdelayReqInterval (minimum permitted mean time
  interval between successive Pdelay_Req messages).";
}

leaf version-number {
  type uint8;
  description
  "The PTP version in use on the port.";
}

container transparent-clock-default-ds {
  description
  "The members of the transparentClockDefault data set (see
  IEEE Std 1588-2008 subclause 8.3.2).";

  leaf clock-identity {
    type clock-identity-type;
    config false;
    description
    "The clockIdentity of the transparent clock.";
  }

  leaf number-ports {
    type uint16;
    description
    "The number of PTP ports on the transparent clock.";
  }

  leaf delay-mechanism {
    type delay-mechanism-enumeration;
  }
}
description
  "The propagation delay measuring option
  used by the transparent clock."
)

leaf primary-domain {
  type uint8;
  default 0;
  description
    "The domainNumber of the primary syntonization domain (see
     IEEE Std 1588-2008 subclause 10.1).";
}

list transparent-clock-port-ds-list {
  key "port-number";
  description
    "List of transparentClockPort data sets of the transparent
     clock (see IEEE Std 1588-2008 subclause 8.3.3)."

  leaf port-number {
    type uint16;
    description
      "Port number.
      The data sets (i.e., information model) of IEEE Std
      1588-2008 specify a member
      transparentClockPortDS.portIdentity, which uses a typed
      struct with members clockIdentity and portNumber.

      In this YANG data model, portIdentity is not modeled in
      the transparent-clock-port-ds-list, however, its
      members are provided as follows:
      portIdentity.portNumber is provided as this leaf member
      in transparent-clock-port-ds-list; and
      portIdentity.clockIdentity is provided as the clock-
      identity leaf in transparent-clock-default-ds
      (i.e., ../../../transparent-clock-default-ds/clock-
      identity)."
  }

  leaf log-min-pdelay-req-interval {
    type int8;
    description
      "The logarithm to the base 2 of the
      minPdelayReqInterval (minimum permitted mean time
interval between successive Pdelay_Req messages)."
);

leaf faulty-flag {
  type boolean;
  default false;
  description
    "When set to true, the port is faulty.";
}

leaf peer-mean-path-delay {
  type time-interval-type;
  default 0;
  description
    "An estimate of the current one-way propagation delay
     on the link when the delayMechanism is P2P; otherwise,
     it is zero.";
}

4. Security Considerations

YANG modules are designed to be accessed via the NETCONF protocol
[RFC6241], thus security considerations in [RFC6241] apply here.
Security measures such as using the NETCONF over SSH [RFC6242] and
restricting its use with access control [RFC8341] can further
improve its security, avoid injection attacks and misuse of the
protocol. Furthermore, general security considerations of time
protocols are discussed in [RFC7384].

All subtrees and most data nodes defined in this YANG module are
writable:

/ptp/instance-list specifies an instance (i.e., PTP data sets) for
an OC or BC.

/ptp/transparent-clock-default-ds specifies a default data set for
a TC.

/ptp/transparent-clock-port-ds-list specifies a list of port data
sets for a TC.
An inappropriate use of them may adversely impact a PTP synchronization network. For example, loss of synchronization on a clock, accuracy degradation on a set of clocks, or even break down of a whole synchronization network.

5. IANA Considerations

This document registers the following URI in the "IETF XML registry" [RFC3688]:
Registrant Contact: The IESG
XML: N/A; the requested URI is an XML namespace

This document registers the following YANG module in the "YANG Module Names" registry [RFC6020]:
Name: ietf-ptp
Prefix: ptp
Reference: RFC XXXX

6. References

6.1. Normative References

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


Raphael Jiang, et al Expires January 24, 2019
6.2. Informative References


[RFC8340] Bjorklund, M., and Berger, L., "YANG Tree Diagrams", RFC 8340, March 2018

7. Acknowledgments

The authors would like to thank Tom Petch, Mahesh Jethanandani, Tal Mizrahi, Opher Ronen, Liang Geng, Alex Campbell, Joe Gwinn, John Fletcher, William Zhao and Dave Thaler for their valuable reviews and suggestions, thank Benoît Claise and Radek Krejci for their validation of the YANG module, and thank Jingfei Lv and Zitao Wang for their discussions on IEEE 1588 and YANG respectively.
Appendix A  Transferring YANG Work to IEEE 1588 WG

This Appendix is informational.

This appendix describes a future plan to transition responsibility for IEEE 1588 YANG modules from the IETF TICTOC Working Group (WG) to the IEEE 1588 WG, which develops the time synchronization technology that the YANG modules are designed to manage.

This appendix is forward-looking with regard to future standardization roadmaps in IETF and IEEE. Since those roadmaps cannot be predicted with significant accuracy, this appendix is informational, and it does not specify imperatives or normative specifications of any kind.

The IEEE 1588-2008 YANG module of this standard represents a cooperation between IETF (for YANG) and IEEE (for 1588). For the initial standardization of IEEE-1588 YANG modules, the information model is relatively clear (i.e., IEEE 1588 data sets), but expertise in YANG is required, making IETF an appropriate location for the standards. The TICTOC WG has expertise with IEEE 1588, making it the appropriate location within IETF.

The IEEE 1588 WG anticipates future changes to its standard on an ongoing basis. As IEEE 1588 WG members gain practical expertise with YANG, the IEEE 1588 WG will become more appropriate for standardization of its YANG modules. As the IEEE 1588 standard is revised and/or amended, IEEE 1588 members can more effectively synchronize the revision of this YANG module with future versions of the IEEE 1588 standard.

This appendix is meant to establish some clear expectations between IETF and IEEE about the future transfer of IEEE 1588 YANG modules to the IEEE 1588 WG. The goal is to assist in making the future transfer as smooth as possible. As the transfer takes place, some case-by-case situations are likely to arise, which can be handled by discussion on the IETF TICTOC WG mailing lists and/or appropriate liaisons.

This appendix obtained insight from [RFC4663], an informational memo that described a similar transfer of MIB work from the IETF Bridge MIB WG to the IEEE 802.1 WG.
A.1. Assumptions for the Transfer

For the purposes of discussion in this appendix, assume that the IESG has approved the publication of an RFC containing a YANG module for a published IEEE 1588 standard. As of this writing, this is IEEE Std 1588-2008, but it is possible that YANG modules for subsequent 1588 revisions could be published from the IETF TICTOC WG. For discussion in this appendix, we use the phrase "last IETF 1588 YANG" to refer to the most recently published 1588 YANG module from the IETF TICTOC WG.

The IEEE-SA Standards Board New Standards Committee (NesCom) handles new Project Authorization Requests (PARs) (see http://standards.ieee.org/board/nes/). PARs are roughly the equivalent of IETF Working Group Charters and include information concerning the scope, purpose, and justification for standardization projects.

Assume that IEEE 1588 has an approved PAR that explicitly specifies development of a YANG module. The transfer of YANG work will occur in the context of this IEEE 1588 PAR. For discussion in this appendix, we use the phrase "first IEEE 1588 YANG" to refer to the first IEEE 1588 standard for YANG.

Assume that as part of the transfer of YANG work, the IETF TICTOC WG agrees to cease all work on standard YANG modules for IEEE 1588.

Assume that the IEEE 1588 WG has participated in the development of the last IETF 1588 YANG module, such that the first IEEE 1588 YANG module will effectively be a revision of it. In other words, the transfer of YANG work will be relatively clean.

The actual conditions for the future transfer can be such that the preceding assumptions do not hold. Exceptions to the assumptions will need to be addressed on a case-by-case basis at the time of the transfer. This appendix describes topics that can be addressed based on the preceding assumptions.

A.2. Intellectual Property Considerations

During review of the legal issues associated with transferring Bridge MIB WG documents to the IEEE 802.1 WG (Section 3.1 and Section 9 of [RFC4663]), it was concluded that the IETF does not have sufficient legal authority to make the transfer to IEEE without the consent of the document authors.
If the last IETF 1588 YANG is published as a RFC, the work is required to be transferred from the IETF to the IEEE, so that IEEE 1588 WG can begin working on the first IEEE 1588 YANG.

When work on the first IEEE YANG module begins in the IEEE 1588 WG, that work derives from the last IETF YANG module of this RFC, requiring a transfer of that work from the IETF to the IEEE. In order to avoid having the transfer of that work be dependent on the availability of this RFC’s authors at the time of its publication, the IEEE Standards Association department of Risk Management and Licensing provided the appropriate forms and mechanisms for this document’s authors to assign a non-exclusive license for IEEE to create derivative works from this document. Those IEEE forms and mechanisms will be updated as needed during the development of this document (Note: i.e., RFC XXXX, update it to the actual RFC if published) and any future IETF YANG modules for IEEE 1588. This will help to make the future transfer of work from IETF to IEEE occur as smoothly as possible.

As stated in the initial "Status of this Memo", the YANG module in this document conforms to the provisions of BCP 78. The IETF will retain all the rights granted at the time of publication in the published RFCs.

A.3. Namespace and Module Name

As specified in Section 5 "IANA Considerations", the YANG module in this document uses IETF as the root of its URN namespace and YANG module name.

Use of IETF as the root of these names implies that the YANG module is standardized in a Working Group of IETF, using the IETF processes. If the IEEE 1588 Working Group were to continue using these names rooted in IETF, the IEEE 1588 YANG standardization would need to continue in the IETF. The goal of transferring the YANG work is to avoid this sort of dependency between standards organizations.

IEEE 802 has an active PAR (IEEE P802d) for creating a URN namespace for IEEE use (see http://standards.ieee.org/develop/project/802d.html). It is likely that this IEEE 802 PAR will be approved and published prior to the transfer of YANG work to the IEEE 1588 WG. If so, the IEEE 1588 WG can use the IEEE URN namespace for the first IEEE 1588 YANG module, such as:

\[ urn:ieee:Std:1588:yang:ieee1588-ptp \]
where "ieee1588-ptp" is the registered YANG module name in the IEEE.

Under the assumptions of section A.1, the first IEEE 1588 YANG module prefix can be the same as the last IETF 1588 YANG module prefix (i.e. "ptp"), since the nodes within both YANG modules are compatible.

The result of these name changes are that for complete compatibility, a server (i.e., IEEE 1588 node) can choose to implement a YANG module for the last IETF 1588 YANG module (with IETF root) as well as the first IEEE 1588 YANG module (with IEEE root). Since the content of the YANG module transferred are the same, the server implementation is effectively common for both.

From a client’s perspective, a client of the last IETF 1588 YANG module (or earlier) looks for the IETF-rooted module name; and a client of the first IEEE 1588 YANG module (or later) looks for the IEEE-rooted module name.

A.4. IEEE 1588 YANG Modules in ASCII Format

Although IEEE 1588 can certainly decide to publish YANG modules only in the PDF format that they use for their standard documents, without publishing an ASCII version, most network management systems cannot import the YANG module directly from the PDF. Thus, not publishing an ASCII version of the YANG module would negatively impact implementers and deployers of YANG modules and would make potential IETF reviews of YANG modules more difficult.

This appendix recommends that the IEEE 1588 WG consider future plans for:

- Public availability of the ASCII YANG modules during project development. These ASCII files allow IETF participants to access these documents for pre-standard review purposes.

- Public availability of the YANG portion of published IEEE 1588 standards, provided as an ASCII file for each YANG module. These ASCII files are intended for use of the published IEEE 1588 standard.

As an example of public availability during project development, IEEE 802 uses the same repository that IETF uses for YANG module development (see https://github.com/YangModels/yang). IEEE branches are provided for experimental work (i.e. pre-PAR) as well as standard work (post-PAR drafts). IEEE-SA has approved use of this repository for project development, but not for published standards.
As an example of public availability of YANG modules for published standards, IEEE 802.1 provides a public list of ASCII files for MIB (see http://www.ieee802.org/1/files/public/MIBs/ and http://www.ieee802.org/1/pages/MIBS.html), and analogous lists are planned for IEEE 802.1 YANG files.
Authors’ Addresses

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

Xian Liu
Independent
Shenzhen 518129, China
lene.liuxian@foxmail.com

Jinchun Xu
Huawei Technologies Co., Ltd.
Bantian, Longgang district
Shenzhen 518129, China
xujinchun@huawei.com

Rodney Cummings (Editor)
National Instruments
11500 N. Mopac Expwy
Bldg. C
Austin, TX 78759-3504
Email: Rodney.Cummings@ni.com
Enterprise Profile for the Precision Time Protocol
With Mixed Multicast and Unicast Messages

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

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

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

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt
The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html
This Internet-Draft will expire on January 31, 2019.

Copyright Notice
Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

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

1. Introduction

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

The "Best Master Clock Algorithm" ([IEEE1588] Subclause 9.3), a mechanism that all participating PTP nodes must follow, set up strict rules for all members of a PTP domain to determine which node shall be the active sending time source (Master Clock).

Although the multicast communication model has advantages in smaller networks, it complicated the application of PTP in larger networks, for example in environments like IP based telecommunication networks or financial data centers. It is considered inefficient that, even if the content of a message applies only to one receiver, it is forwarded by the underlying...
network (IP) to all nodes, requiring them to spend network bandwidth and other resources, such as CPU cycles, to drop the message.

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

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

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119].

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

3. Technical Terms

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

Alternate Master: A PTP Master Clock, which is not the Best Master, may act as a master with the Alternate Master flag set on the messages it sends.
Announce message: Contains the Master Clock properties of a Master Clock. Used to determine the Best Master.

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

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

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

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

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

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

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

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

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

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

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

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

Preferred Master: A device intended to act primarily as the Grandmaster of a PTP system, or as a back up to a Grandmaster.
PTP: The Precision Time Protocol, the timing and synchronization protocol defined by IEEE 1588.

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

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

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

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

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

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

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

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

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

4. Problem Statement

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

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

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

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

5. Network Technology

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

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

PTP, like NTP, assumes that the one-way network delay for Sync Messages and Delay Response Messages are the same. When this is not true it can cause errors in the transfer of time from the Master to the Slave. It is up to the system integrator to design the network so that such effects do not prevent the PTP system from meeting the timing requirements. The details of
network asymmetry are outside the scope of this document. See for example, [G8271].

6. Time Transfer and Delay Measurement

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

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

Sync messages MUST be sent as PTP event multicast messages (UDP port 319) to the PTP primary IP address. Two step clocks SHALL send Follow-up messages as PTP general messages (UDP port 320). Announce messages MUST be sent as multicast messages (UDP port 320) to the PTP primary address. The PTP primary IP address is 224.0.1.129 for IPv4 and FF0X:0:0:0:0:0:181 for IPv6, where X can be a value between 0x0 and 0xF, see [IEEE1588] Annex E, Section E.3.

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

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

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

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

8. Requirements for Master Clocks

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

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

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

9. Requirements for Slave Clocks

Slave clocks MUST be able to operate properly in a network which contains multiple Masters in multiple domains. Slaves SHOULD make use of information from the all Masters in their clock control subsystems. Slave Clocks MUST be able to operate properly in the presence of a Rogue Master. Slaves SHOULD NOT Synchronize to a Master which is not the Best Master in its domain. Slaves will continue to recognize a Best Master for the duration of the Announce Time Out Interval. Slaves MAY use an Acceptable Master Table. If a Master is not an Acceptable Master, then the Slave MUST NOT synchronize to it. Note that IEEE 1588-2008 requires slave clocks to support both two-step or one-step Master clocks. See [IEEE1588], subClause 11.2.
Since Announce messages are sent as multicast messages slaves can obtain the IP addresses of a master from the Announce messages. Note that the IP source addresses of Sync and Follow-up messages may have been replaced by the source addresses of a Transparent Clock, so, slaves MUST send Delay Request messages to the IP address in the Announce message. Sync and Follow-up messages can be correlated with the Announce message using the clock ID, which is never altered by Transparent Clocks in this profile.

10. Requirements for Transparent Clocks

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

11. Requirements for Boundary Clocks

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

12. Management and Signaling Messages

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

13. Forbidden PTP Options

Clocks operating in the Enterprise Profile SHALL NOT use peer to peer timing for delay measurement. Grandmaster Clusters are NOT ALLOWED. The Alternate Master option is also NOT ALLOWED. Clocks operating in the Enterprise Profile SHALL NOT use Alternate Timescales. Unicast discovery and unicast negotiation SHALL NOT be used.
14. Interoperation with IEEE 1588 Default Profile

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

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

15. Profile Identification

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

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

This profile was specified by the IETF

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

16. Security Considerations

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

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

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

17. IANA Considerations

There are no IANA requirements in this specification.
18. References

18.1. Normative References


18.2. Informative References


19. Acknowledgments

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

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

20. Authors’ Addresses

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

Email: doug.arnold@meinberg-usa.com

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

Email: Heiko.gerstung@meinberg.de
NTP Correction Field
draft-mlichvar-ntp-correction-field-02

Abstract

This document specifies an extension field for the Network Time Protocol (NTP) which improves resolution of specific fields in the NTP header and allows network devices such as switches and routers to modify NTP packets with corrections to improve accuracy of the synchronization in the network.

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 December 20, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Processing and queueing delays in network switches and routers may be a significant source of jitter and asymmetry in network delay, which has a negative impact on accuracy and stability of clocks synchronized by NTP [RFC5905].

If all network devices on the paths between NTP clients and servers implemented NTP and supported an operation as a server and client, the impact of the delays could be avoided by configuring NTP to make measurements only between devices and hosts that are directly connected to one another. In the Precision Time Protocol (PTP) [IEEE1588], which is a different protocol for synchronization of clocks in networks, such devices are called Boundary Clocks (BC).

A different approach supported by PTP to improve the accuracy uses Transparent Clocks (TC). Instead of fully implementing PTP in order to support an operation as a BC, the devices only modify a correction field in forwarded PTP packets with the time that the packets had to wait for transmission. The final value of the correction is included in the calculation of the delay and offset, which may significantly improve the accuracy and stability of the synchronization.

This document describes an NTP extension field which allows the devices to make a similar correction in forwarded NTP packets.

To better support a highly accurate synchronization, the extension field also improves resolution of the following fields in the NTP header: root delay, root dispersion, receive timestamp, transmit timestamp.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Format of Correction Field

The Correction Field is an NTP extension field following RFC 7822 [RFC7822]. The format of the extension field is shown in Figure 1.
The extension field has the following fields:

Field Type
The type which identifies the Correction extension field.
TBD

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

Root Delay Correction
A 16-bit extension of the root delay in the NTP header increasing its resolution. The extended root delay has 16 integer bits and 32 fractional bits.

Root Dispersion Correction
A 16-bit extension of the root dispersion in the NTP header increasing its resolution. The extended root dispersion has 16 integer bits and 32 fractional bits.

Receive Correction
An 8-bit extension of the receive timestamp in the NTP header increasing its resolution. The extended receive timestamp has 32 integer bits and 40 fractional bits.

Transmit Correction
An 8-bit extension of the transmit timestamp in the NTP header increasing its resolution. The extended transmit timestamp has 32 integer bits and 40 fractional bits.
Origin Correction
A field which contains a copy of the final delay correction from the previous packet in the NTP exchange.

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

Delay Correction
A signed fixed-point number of seconds with 8 integer bits and 40 fractional bits, which represents the current correction of the network delay that has accumulated for this packet on the path from the source to the destination.

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

Checksum Complement
A field which can be modified to not change the UDP checksum after updating the other fields. The same field is described in RFC 7821 [RFC7821].

3. Network devices

A network device which is forwarding a packet and supports the Correction Field MUST NOT modify the packet unless all of the following applies:

1. The packet is an IPv4 or IPv6 UDP packet.

2. The source port or destination port is 123.

3. The NTP version is 4.

4. The NTP mode is 1, 2, 3, 4, or 5.

5. The format of the packet is valid per RFC 7822.

6. The packet contains an extension field which has a type of TBD and length of 28 octets.

The device SHOULD add to the current value in the delay correction field the length of an interval between the reception and transmission of the packet. If the packet is transmitted at the same speed as it was received and the length of the packet does not change (e.g. due to adding or removing a VLAN tag), the beginning and end of the interval may correspond to any point of the reception and
transmission as long as it is consistent for all forwarded packets of the same length. If the transmission speed or length of the packet is different, the beginning and end of the interval SHOULD correspond to the end of the reception and beginning of the transmission respectively.

If the transmission starts before the reception ends, a negative value may need to be added to the delay correction. The end of the reception SHOULD be determined using the length field of the UDP header and the speed at which the packet is received.

If the device updates the delay correction, it SHOULD also add the identification numbers of the incoming and outgoing port to the path ID.

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

4. NTP hosts

When an NTP client sends a request to a server and the association is configured to use the Correction Field, it SHOULD add the extension field to the packet. All fields of the extension field except type and length SHOULD be set to zero.

When the server receives a packet which includes the extension field, the response SHOULD also include the extension field.

The server SHOULD use the root delay and root dispersion correction fields to save additional bits of the root delay and dispersion below the resolution of the 32-bit fixed-point fields in the NTP header. These corrections allow the server to provide the client with root delay and root dispersion in resolution of about 1/4 of a nanosecond.

If the server’s clock has a better precision than resolution of the 64-bit NTP timestamp format, the server SHOULD save the additional bits in the receive and transmit correction fields and set the precision field to the corresponding number, which is smaller than -32. Otherwise, the receive and transmit correction fields SHOULD be zero.

The origin correction and origin ID fields SHOULD be set to the delay correction and path ID from the request. The other fields of the Correction Field SHOULD be zero.

When the client receives a response which contains the extension field, it SHOULD check the value of both the origin and delay
correction fields. If a correction is larger than a specified maximum (e.g. 1 second), the extension field SHOULD be ignored.

The client MAY log a warning if the origin ID and path ID are not equal, which indicates the network path between the server and client is not symmetric.

The client SHOULD extend the root delay and root dispersion from the NTP header with bits from the root delay and root dispersion correction fields respectively.

If the client’s clock has a better precision than resolution of the 64-bit NTP format and the precision field in the response contains a number smaller than -32, the client SHOULD extend the receive and transmit timestamp from the NTP header with the additional bits from the receive and transmit correction fields respectively.

When the client computes the offset and delay using the formulas from RFC 5905, the origin correction is subtracted from the receive timestamp and the delay correction is added to the transmit timestamp.

An NTP peer follows the rules of both servers and clients. It processes Correction Fields in received packets as a client and sends Correction Fields as a server. A packet which has a zero origin timestamp (i.e. it is not a response to a request) SHOULD have a zero origin correction and zero origin ID in the Correction Field.

A broadcast server using the Correction Field SHOULD always set the origin correction and origin ID fields to zero.

5. Acknowledgements

The Correction Field extension is based on the PTP correction field specified in IEEE 1588-2008.

The author would like to thank Tal Mizrahi and Harlan Stenn for their useful comments.

6. IANA Considerations

IANA is requested to allocate an Extension Field Type for the Correction Field.
7. Security Considerations

NTP packets including the Correction Field cannot be authenticated by a legacy MAC, because the MAC has to cover all extension fields in the packet and devices which are supposed to modify the field are not able to update the MAC.

It is recommended to authenticate NTP packets using an authentication extension field, e.g. the NTS Authenticator and Encrypted Extensions [I-D.ietf-ntp-using-nts-for-ntp] extension field, and add the Correction Field to the packet after the authentication field.

A man-in-the-middle attacker can delay packets in the network in order to increase the measured delay and shift the measured offset by up to half of the extra delay. If the packets contain the Correction Field, the attacker can reduce the delay calculated by the client or peer and shift the offset even more. The maximum correction should be limited (e.g. to 1 second) to prevent the attacker from injecting a larger offset to the measurements.

8. References

8.1. Normative References


8.2. Informative References


Author’s Address

Miroslav Lichvar
Red Hat
Purkynova 115
Brno 612 00
Czech Republic

Email: mlichvar@redhat.com
NTP Interleaved Modes

draft-mlichvar-ntp-interleaved-modes-01

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 June 15, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must
1. Introduction

RFC 5905 [RFC5905] describes the operations of NTPv4 in basic client/server, symmetric, and broadcast mode. The transmit timestamp is one of the four timestamps included in every NTP packet used for time synchronization. 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.

There are, at least, four options where a transmit timestamp can be captured i.e. by NTP daemon, by network drivers, or at the MAC or physical layer of the OSI model. A typical transmit timestamp in a software NTP implementation in the basic mode is the one captured by the NTP daemon using the system clock, before the computation of message digest and before the packet is passed to the operating system, and does not include any processing and queuing delays in the system, network drivers, and hardware. These delays may add a significant error to the offset and network delay measured by clients and peers of the server.

For best accuracy, the transmit timestamp should be captured as close to the wire as possible, but that is difficult to implement in the current packet since this timestamp is available only after the packet transmission. The protocol described in RFC 5905 does not specify any mechanism for the server to provide its clients and peers with this more accurate timestamp.

Different mechanisms could be used to exchange this more accurate timestamp. This document describes interleaved modes, in which an NTP packet contains a transmit timestamp corresponding to the previous packet that was sent to the client or peer. This transmit timestamp could be captured at one of the any four places mentioned above. More specifically, this document:

1. Introduces and specifies a new interleaved client/server mode.
2. Specifies the interleaved symmetric mode based on the NTP reference implementation with some modifications.
3. Specifies the interleaved broadcast mode based purely on the NTP reference implementation.

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.

The protocol requires both servers and clients/peers to keep some state specific to the interleaved mode. It prevents traffic amplification that would be possible if the timestamp was sent in a separate message in order to keep the servers stateless.

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", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

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.

When the server receives a request, it SHOULD compare the origin timestamp with all receive timestamps it has saved (for the IP address). If a match is found, the server SHOULD respond with a packet in the interleaved mode, which contains the transmit timestamp
corresponding to the packet which had the matching receive timestamp. If no match is found, the server MUST NOT respond in the interleaved mode. The server MAY always respond in the basic mode. In both cases, the server SHOULD save the new receive and transmit timestamps.

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 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 client SHOULD limit the number of requests in the interleaved mode per server response to prevent processing of very old timestamps in case a large number of packets 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).

```
Server   t2   t3               t6   t7              t10  t11
-----+----+----------------+----+----------------+----+-----
     /      \              /      \              /      \
Client /        \            /        \            /        \
-----+----------+----------+----------+----------+----------+-----
Org 0 | t1′       | t2       | t4       | t6       | t5       |
Rx 0  | t2         | t4       | t6       | t8       | t10      |
Tx t1′ | t3′       | t1       | t3       | t5       | t11′      |
```
When the client receives a response, it performs all tests described in RFC 5905, except now the sanity check for bogus packet needs to compare the origin timestamp with both transmit and receive timestamps from the request in order to be able to detect if the response is in the basic or 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 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)initiation 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 of the two sets of timestamps specified in the client/server section. They MAY switch between them to minimize the interval between \( T_1 \) and \( T_4 \) 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.
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 reference NTP implementation written by David Mills.

The authors would like to thank Kristof Teichel for his useful comments.

6. IANA Considerations

This memo includes no request to IANA.

7. Security Considerations

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 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.

Protecting symmetric associations in the interleaved mode against replay attacks is even more difficult than in the basic mode, because 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

[RFC2119]  Bradner, S., "Key words for use in RFCs to Indicate 
Requirement Levels", BCP 14, RFC 2119, 
DOI 10.17487/RFC2119, March 1997, 

Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, 

8.2. Informative References

[I-D.ietf-ntp-data-minimization] 
Franke, D. and A. Malhotra, "NTP Client Data 
Minimization", draft-ietf-ntp-data-minimization-01 (work 
in progress), July 2017.

[SECNTP] Malhotra, A., Gundy, M., Varia, M., Kennedy, H., Gardner, 
J., and S. Goldberg, "The Security of NTP's Datagram 

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

Lichvar & Malhotra Expires June 15, 2018 [Page 9]
Network Time Protocol Version 4 (NTPv4) Extension Fields
draft-stenn-ntp-extension-fields-06

Abstract

Network Time Protocol version 4 (NTPv4) defines the optional usage of extension fields. An extension field, as defined in RFC 5905 [RFC5905] and RFC 5906 [RFC5906], resides after the end of the NTP header and supplies optional capabilities or information that is not conveyed in the standard NTP header. This document updates RFC 5905 [RFC5905] by clarifying some points regarding NTP extension fields and their usage with legacy Message Authentication Codes (MACs).

This proposal deprecates RFC 7822 [RFC7822].

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 September 22, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents.
1. Introduction

An NTP packet consists of a set of fixed fields that may be followed by optional fields. Two types of optional fields are defined: extension fields (EFs) as defined in Section 7.5 of RFC 5905 [RFC5905], and legacy Message Authentication Codes (legacy MACs).

If a legacy MAC is used, it resides at the end of the packet. This field can be either a 4-octet crypto-NAK or data that has traditionally been 16, 20 or 24 octets long.

Additional information about the content of a MAC is specified in RFC 5906 [RFC5906], but since that RFC is Informational an implementor that was not planning to provide Autokey would likely never read that document. The result of this would be interoperability problems, at least. To address this problem, this proposal also includes copying and clarifying some of the content of RFC 5906 and putting it into RFC 5905. Because there is a reasonable expectation that RFC 5906...
will be deprecated, this document does not propose changes or updates to RFC 5906.

NTP extension fields are defined in RFC 5905 [RFC5905] as a generic mechanism that allows the addition of future extensions and features without modifying the NTP header format (Section 16 of RFC 5905 [RFC5905]).

With the knowledge and experience we have gained over time, it has become clear that simplifications, clarifications, and improvements can be made to the NTP specification around EFs and MACs.

This proposal adjusts the requirements around EFs and MACs, allows EFs to be on 4-octet boundaries of any acceptable length, and provides methods to disambiguate packet parsing in the unexpected and unlikely case where an implementation would choose to send a packet that could be ambiguously parsed by the receiver.

This proposal deprecates RFC 7822 [RFC7822].

This document better specifies and clarifies extension fields as well as the requirements and parsing of a legacy MAC, with changes to address errors found after the publication of RFC 5905 [RFC5905] with respect to extension fields. Specifically, this document updates Section 7.5 of RFC 5905 [RFC5905], clarifying the relationship between extension fields and MACs, and expressly defines the behavior of a host that receives an unknown extension field.

2. Conventions Used in This Document

2.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2.2. Terms and Abbreviations

EF - Extension Field

MAC - Message Authentication Code

NTPv4 - Network Time Protocol, Version 4 RFC 5905 [RFC5905]
3. NTP MAC - RFC 5906 Update

This document copies and updates some information in RFC 5906 [RFC5906] and puts it in to RFC 5905, as follows:

3.1. RFC5906 Section 4. - Autokey Cryptography

This section describes some of the cryptography aspects of Autokey. The third paragraph describes the use of 128- and 160-bit message digests. The enumeration of 128- and 160-bit message digests is not meant to be limiting - other message digest lengths MAY be implemented. This paragraph also describes some of the recommended semantic ranges of the key ID. This information belongs in RFC 5905. The key ID value is particularly significant because it provide additional disambiguation protection when deciding if the next data portion is either a legacy MAC or an extension field.

3.2. RFC5906 Section 10. - Autokey Protocol Messages

This section describes the extension field format, including initial flag bits, a Code field, and 8-bit Field Type, and the 16-bit Length. This proposal expands and clarifies this information and puts it into RFC 5905.

This section says "The reference implementation discards any packet with a field length of more than 1024 characters." but this is no longer true.

3.3. RFC5906 Section 11.5. - Error Recovery

This section describes the crypto-NAK, which should be described in RFC 5905.

3.4. RFC5906 Section 13. - IANA Consideration

This section lists the Autokey-related Extension Field Types, including Flag Bits, Codes, and Field Types, which should be described in RFC 5905, or perhaps in some other document.

4. NTP Extension Fields - RFC 5905 Update

This document updates Section 7.5 of RFC 5905 [RFC5905] as follows:

4.1. OLD: RFC5905 7.5 - NTP Extension Field Format

In NTPv4, one or more extension fields can be inserted after the header and before the MAC, which is always present when an extension field is present. Other than defining the field format, this
document makes no use of the field contents. An extension field contains a request or response message in the format shown in Figure 14.

```
+---------------+---------------+-------------------------------+
|          Field Type           |        Field Length           |
+-------------------------------+-------------------------------+
|                       Padding (as needed)                     |
+---------------------------------------------------------------+
```

Figure 14: Extension Field Format

All extension fields are zero-padded to a word (four octets) boundary. The Field Type field is specific to the defined function and is not elaborated here. While the minimum field length containing required fields is four words (16 octets), a maximum field length remains to be established.

The Length field is a 16-bit unsigned integer that indicates the length of the entire extension field in octets, including the Padding field.

4.2. NEW: RFC5905 Section 7.5 - NTP Extension Field Format

In NTPv4, one or more extension fields can be inserted after the header and before the possibly optional legacy MAC. A MAC SHOULD be present when an extension field is present. A MAC is always present in some form when NTP packets are authenticated. This MAC SHOULD be either a legacy MAC or a MAC-EF. It MAY be both. Other than defining the field format, this document makes no use of the field contents. An extension field contains a request or response message in the format shown in Figure 14.
All extension fields are zero-padded to a word (four octet) boundary. The Field Type is specific to the defined function and detailed information about the Field Type is not elaborated here. The minimum size of an Extension Field is a 32-bit word (4 octets), and while the maximum extension field size MUST be 65532 octets or less, an NTP packet SHOULD NOT exceed the network MTU.

The Length field is a 16-bit unsigned integer that indicates the length of the entire extension field in octets, including any Padding octets. The bottom two bits of the Field Length SHOULD be zero, and the size of the extension field SHOULD end on a 32-bit (4 octet) boundary. [RFC5905 Section 7.5 says "All extension fields are zero-padded to a word (four octets) boundary." but does not use 'MUST' language. Is it overkill to reiterate this requirement here? Should we use SHOULD or MUST regarding the bottom two bits or the boundary of the EF? It is possible, down the road, that we might find some use for those bottom 2 bits, even if we require a 32-bit boundary on the last octet of an EF.]

The Field Type contains the following sub-elements:

<table>
<thead>
<tr>
<th>R</th>
<th>E</th>
<th>Code</th>
<th>Type</th>
<th>(Field Length)</th>
</tr>
</thead>
</table>

Field Type Format

Where the following Field Type flags are defined:

R: 0 for a "Query", 1 for a "Response"

E: 0 for "OK", 1 for an "Error". Unused, and will be deprecated.
[The ‘R’ flag is currently used by Autokey, and by the proposed I-DO extension field. This flag is used after the packet is accepted.]

[The ‘E’ flag was proposed for use by Autokey, after the packet was accepted. As it was never used and no other use-cases have been identified, we are recommending this flag be deprecated at some point in the future.]

[The EF Code subtype is currently used by RFC 5906, Autokey [RFC5906]. The EF Code subtype is used by the proposed Extended Information EF proposal, and is expected to be used by the NTS Extension Field, at least.]

The Autokey EF currently uses the most Code values - 10 of them, which equates to the least-significant 4 bits of the high-order octet. It is possible that additional flag bits will be allocated; in the past, the high-order 2 bits were reserved, and for a time two additional bits were proposed. Make no assumptions about the unused bits in this octet.

The Field Type, Value, and Padding fields are specific to the defined function and are not elaborated here; appropriate Field Type flags, the EF Code, and EF Type values are defined in an IANA registry, and the Length, Value, and Padding values are defined by the document referred to by the registry. If a host receives an extension field with an unknown Field Type, the host SHOULD ignore the extension field and MAY drop the packet altogether, depending on local policy.

The Length field is a 16-bit unsigned integer that indicates the length of the entire extension field in octets, including any Padding.

While the minimum field length of an EF that contains no value fields is one word (four octets), and the minimum field length of an EF that contains required fields is two words (8 octets), the maximum field length MUST NOT be longer than 65532 octets due to the maximum size of the data represented by the Length field, and SHOULD be small enough that the size of the NTP packet received by the client does not exceed the smallest MTU between the sender and the recipient. The bottom two bits of the Field Length SHOULD be zero and the EF data SHOULD be aligned to a 32-bit (4 octet) boundary.

4.3. NEW: RFC5905 Section 7.5.1 - Extension Fields and MACs

With the inclusion of additional Extension Fields, there is now a potential that a poorly-designed implementation would produce an ambiguous parsing in the presence of a legacy MAC. If an implementation offers even a modicum of care, there will be no
ambiguity when parsing an NTP packet that contains a legacy MAC from an existing implementation.

The first protection from this ambiguity comes from the fact that current conforming implementations only support the Autokey EF, which uses EF Type 2 and a legacy MAC. While the Experimental UDP Checksum Complement specified by RFC 7821 [RFC7821] uses EF Type 5, it specifically prohibits the use of a MAC, and the 0x2000 bit in its assigned EF specification of 0x2005 originally signified that a MAC is optional when this EF is seen. While the 0x2000 bit is no longer proposed as a means to flag that the MAC is optional, any usage of this EF with a Code field of either 0x2005 or 0x0005 can be trivially recognized as an Experimental UDP Checksum Complement EF, which does not, indeed, MUST NOT be followed by a MAC.

[As a side note, the requirement in RFC 7821 [RFC7821] that the UDP Checksum Complement EF must have a 28 octet length is demonstrably not needed if this proposal is accepted. It only needs 8 octets: 4 octets of EF header, 2 octets of must-be-zero padding, and 2 octets of Checksum Complement.]

If an implementation uses the LAST-EF extension field, the presence of this field means "I am the last EF in this NTP Packet. Any subsequent packet data MUST be a legacy MAC." In this case, there is no parsing ambiguity.

If a system sends its MAC as a MAC-EF and does not send a legacy MAC, there is no parsing ambiguity.

The only time there is a potential for a parsing ambiguity is when a legacy MAC is provided and neither of the previous two cases are present. Even in this case, there is minimal risk.

An implementation MAY choose to add padding to any EF, or at least any EF that comes before a legacy MAC, to avoid an EF that is 16, 20, or 24 octets in length. Doing this would generally avoid any risk of mis-parsing. But this should not be needed for the following reasons.

An Extension Field contains a 2-octet Field Type, a 2-octet Field Length, and any payload (data and/or padding). If the NTP Packet parsing is at a point where it is evaluating data after the base packet, one of the following situations exists:

- If the Field Length is not an even multiple of 4, we are not looking at an extension field. In this case, the only possibility of having a valid packet is if the data is part of a legacy MAC.
If the Field Length is valid, i.e., an even multiple of 4 octets, one of the following three cases must be present:

First, the Field Length will be less than the remaining data. This means subsequent data must parse as some number of Extension Fields, optionally followed by a legacy MAC.

Second, the Field Length will exactly match the remaining data.

The third case is where the Field Length is longer than the remaining packet data. In this case, the current parse cannot be a valid extension field, and if the packet is valid, the data must be a legacy MAC.

Semantic checking may also be done to validate a potential legacy MAC. A legacy MAC is a four-octet Key Identifier followed by a message digest. The usual message digest is 16 octets long but may be another size, depending on the digest algorithm. In the Reference Implementation and in implementations that follow the guidelines for the values of the Key Identifier, a Key Identifier between 1 and 65535, inclusive, is a symmetric key, while a Key Identifier that is > 65535 is an Autokey RFC 5906 [RFC5906], or similar. If the receiving system does not recognize the Key Identifier, the data CANNOT be a valid legacy MAC. If the receiving system recognizes the Key Identifier, then it also has knowledge of the digest algorithm and can make sure the digest payload is the proper length. If this is not the case, then the data CANNOT be a valid legacy MAC. In this case, it MIGHT be a valid extension field.

It is trivial to parse the data after the base NTP packet and come up with a list of potential parsings. A local policy choice can specify the precedence of the parsing options in this case.

If none of the parsings validate, the packet fails authentication. An implementation has three local policy choices available if LAST-EF is not used and a legacy MAC may be provided. First, the implementation may specify EF-precedence. Second, the implementation may specify legacy-MAC-precedence. Finally, the implementation may specify "best fit" precedence. In this last case, the packet will meet one of the three following criteria: First, none of the parsings will match. Again, this is a case of failed authentication. Second, exactly one parsing will match and that parsing will be accepted. Third, multiple parsings will match, in which case the implementation may choose its behavior.

Additionally, most EFs will require a MAC. If there is a syntactically-valid parsing that does not include a MAC but previously scanned EFs require a MAC, then in a multiple-choice
parsing scenario where one of the choices does not include a MAC the "no MAC provided" choice SHOULD be eliminated.

Note well that this rare situation can be completely avoided by using LAST-EF, or by indicating that no legacy MAC will be used.

Finally, in many cases at least one side will know if a MAC is required or not. Client configurations of all types, unicast, broadcast, multicast, and manycast, that state that a key will be used to communicate with a server SHOULD reject packets claiming to be from the server that do not include a MAC. Symmetric associations also are configured with similar knowledge and requirements.

4.3.1. Legacy MAC/EF Parsing Pseudocode

Here are two potential pseudocode implementations showing how data after the base NTP packet could be analyzed to identify EFs and a possible legacy MAC.

Example 1: Generate a list of possible parsings:
struct pkt_parse {
    foo * ef_ptr;
    foo * legacy_mac;
    struct pkt_parse * next;
};

struct pkt_parse pkt_parse_chain = NULL;

EOPacket = address of last data in packet;
here = address of the EOBasePacket;
more_efs = 1;
while (1) {
    int candidate = 0;
    int ef_len = 0;
    
    if (EOPacket > here) {
        p = emalloc(pkt_parse);                  // *p is zeroed
        if (this could be a legacy MAC) {        // we know the keyid
            p->legacy_mac = here;
            candidate = 1;
        }
        if (more_efs && this could be an EF) {  // Length field valid
            p->ef_ptr = here;
            ef_len = (the length of the EF);
            here += ef_len;
            if (this is a LAST_EF) {
                more_efs = 0;
            } else {
                more_efs = 0;
            }
        }
    }
    if (candidate) {
        p->next = pkt_parse_chain;
        pkt_parse_chain = p;
    } else {
        free(p);
        break;
    }
}

Example 1: Generate a list of possible parsings
and at this point we can scan thru the items in pkt_parse_chain to do
deeper checks, throwing away the parsings that don't make sense.
This opens up more questions if we get multiple parsings and at least 1 of them is "valid". It’s also perfectly reasonable to decide to produce a single parse based on precedence rules: Prefer legacy MAC, or prefer EF.

Example 2: Another possible way to handle EF/legacy-MAC parsing:

```c
ef_ok = 1;                              // An EF is allowed here
legacy_mac_ok = 1;                      // Legacy MAC allowed here
saw_mac = 0;                            // We haven’t seen a MAC yet
authlen = LEN_PKT_NOMAC;                // Length of a base packet
leg_mac = rbufp->recv_length - authlen; // # bytes after base
while (leg_mac > 0) {                   // Data after base packet
    if (leg_mac % 4 != 0 || leg_mac < MIN_MAC_LEN) {
        return: Bad packet length;
    }

    // If ef_ok, this could be an EF or legacy MAC
    skeyid = ntohl(pkt[authlen / 4]);
    opcode = skeyid >> 16;
    len = skeyid & 0xffff;

    if (ef_ok && GET_EXT_FIELD_TYPE(opcode) == EF_FT_LAST) {
        if (leg_mac > MAX_MAC_LEN) {
            return: Too much data after LAST_EF;
        }
        // Anything here MUST be a legacy MAC
        ef_ok = 0;
        legacy_mac_ok = 1;
    } else {
        if (4 == leg_mac && 0 == skeyid) {
            break; // Likely crypto-NAK
        }

        if (legacy_mac_ok && leg_mac <= MAX_MAC_LEN) {
            int ksize;

            // If we find a keyid, we know its alg/length
            ksize = auth_findkeysizes(skeyid);
            if (ksize != -1
                && ksize == leg_mac
```
&& (it validates)) {
    saw_mac = 1;
    break;
}
// If we got here, it can’t be a valid
// legacy MAC. It’s still a potential EF.
}
if (!ef_ok) {
    break;
}
// At this point, this SHOULD be an EF
if (len % 4 != 0
    len < 4
    || len + authlen > rbufp->recv_length) {
    return: Bad length;
}
switch (GET_EXT_FIELD_TYPE(opcode)) {
    case EF_FT_AK: // Autokey
        // extract calling group name for later
        break;
    case EF_FT_LAST: // LAST-EF
        legacy_mac_ok = 1;
        break;
    default:
        legacy_mac_ok = 0;
        break;
}
    authlen += len;
    leg_mac -= len;
}
if (leg_mac < 0) {
    return: Malformed packet
}

Example 2: Another way to handle EF/legacy-MAC parsing

4.4. OLD: RFC5905 Section 9.2. - Peer Process Operations

...
FXMIT. ... This message includes the normal NTP header data shown in Figure 8, but with a MAC consisting of four octets of zeros. ...

4.5. NEW: RFC5905 Section 9.2. - Peer Process Operations

... 

FXMIT. ... This message includes the normal NTP header data shown in Figure 8, but with a MAC consisting of four octets of zeros. This MAC can be a legacy MAC or a MAC-EF. If it’s a MAC-EF, the crypto-NAK MUST be the only MAC in the MAC-EF payload. ...

5. Acknowledgements

The authors wish to acknowledge the contributions of Sam Weiler, Danny Mayer, and Tal Mizrahi.

6. IANA Considerations

This memo requests IANA to allocate the following bits in the NTP Extension Field Types table:

- 0x8000: R: Response (0: Request, 1: Response)
- 0x4000: E: Error (0: OK, 1: Error) - Unused, deprecation expected

The following table should be the functionally the same as the existing NTP Extension Field Table.
### Field Type | Meaning
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>crypto-NAK (with Field Length of 0)</td>
</tr>
<tr>
<td>0x0000</td>
<td>RESERVED: Permanently Unassigned</td>
</tr>
<tr>
<td>0x0001</td>
<td>RESERVED: Unassigned</td>
</tr>
<tr>
<td>0x0002</td>
<td>Autokey: No-Operation Request</td>
</tr>
<tr>
<td>0x0002</td>
<td>Autokey: No-Operation Response</td>
</tr>
<tr>
<td>0x0102</td>
<td>Autokey: Association Message Request</td>
</tr>
<tr>
<td>0x0102</td>
<td>Autokey: Association Message Response</td>
</tr>
<tr>
<td>0x0202</td>
<td>Autokey: Certificate Message Request</td>
</tr>
<tr>
<td>0x0202</td>
<td>Autokey: Certificate Message Response</td>
</tr>
<tr>
<td>0x0302</td>
<td>Autokey: Cookie Message Request</td>
</tr>
<tr>
<td>0x0302</td>
<td>Autokey: Cookie Message Response</td>
</tr>
<tr>
<td>0x0402</td>
<td>Autokey: Autokey Message Request</td>
</tr>
<tr>
<td>0x0402</td>
<td>Autokey: Autokey Message Response</td>
</tr>
<tr>
<td>0x0502</td>
<td>Autokey: Leapseconds Value Message Request</td>
</tr>
<tr>
<td>0x0502</td>
<td>Autokey: Leapseconds Value Message Response</td>
</tr>
<tr>
<td>0x0602</td>
<td>Autokey: Sign Message Request</td>
</tr>
<tr>
<td>0x0602</td>
<td>Autokey: Sign Message Response</td>
</tr>
<tr>
<td>0x0702</td>
<td>Autokey: IFF Identity Message Request</td>
</tr>
<tr>
<td>0x0702</td>
<td>Autokey: IFF Identity Message Response</td>
</tr>
<tr>
<td>0x0802</td>
<td>Autokey: GQ Identity Message Request</td>
</tr>
<tr>
<td>0x0802</td>
<td>Autokey: GQ Identity Message Response</td>
</tr>
<tr>
<td>0x0902</td>
<td>Autokey: MV Identity Message Request</td>
</tr>
<tr>
<td>0x0902</td>
<td>Autokey: MV Identity Message Response</td>
</tr>
<tr>
<td>0x0005</td>
<td>Checksum Complement</td>
</tr>
<tr>
<td>0x1005</td>
<td>Checksum Complement</td>
</tr>
</tbody>
</table>

**Current Extension Fields**

7. Security Considerations

Additional information TBD

8. Normative References


Authors' Addresses

Harlan Stenn
Network Time Foundation
P.O. Box 918
Talent, OR 97540
US

Email: stenn@nwtime.org

David L. Mills
Network Time Foundation
P.O. Box 918
Talent, OR 97540
US

Email: mills@udel.edu